▼ AMSTRAD ▼ SPECTRUM ▼ COMMODORE ▼ IBM ▼ MSX ▼

Informática y programación 17

PASO A PASO



PROGRAMAS EDUCATIVOS
PROGRAMAS DE UTILIDAD
PROGRAMAS DE GESTION
PROGRAMAS DE JUEGOS

▼ BASIC ▼ MAQUINA ▼ PASCAL ▼ LOGO ▼ OTROS LENGUAJES ▼

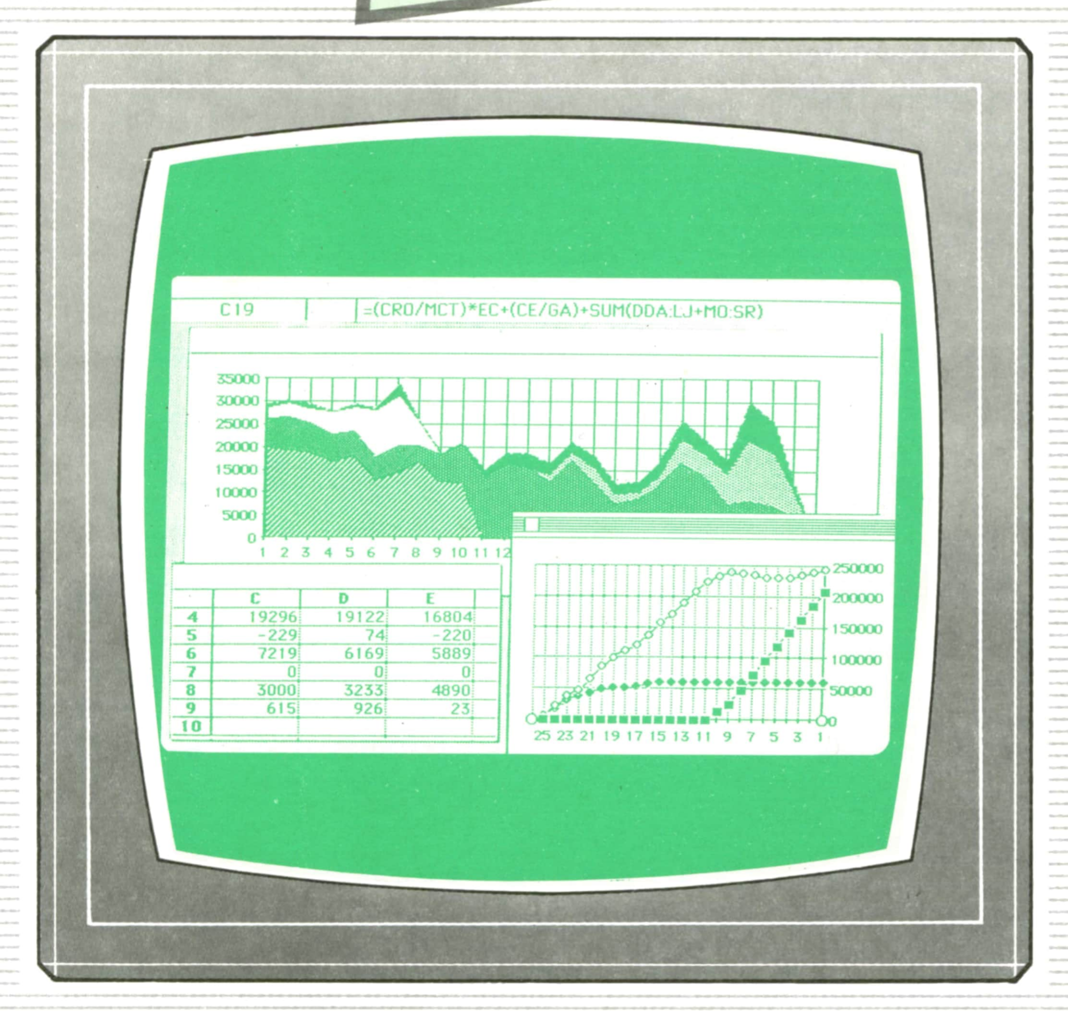
▼ TECNICAS DE ANALISIS Y DE PROGRAMACION ▼

▼ EDICIONES ▼ SIGLO ▼ CULTURAL ▼

Informática y programación

17

PASO A PASO



PROGRAMAS EDUCATIVOS
PROGRAMAS DE UTILIDAD
PROGRAMAS DE GESTION
PROGRAMAS DE JUEGOS

▼ BASIC ▼ MAQUINA ▼ PASCAL ▼ LOGO ▼ OTROS LENGUAJES ▼
▼ TECNICAS DE ANALISIS Y DE PROGRAMACION ▼

▼ EDICIONES ▼ SIGLO ▼ CULTURAL ▼

Una publicación de

EDICIONES SIGLO CULTURAL, S.A.

Director-editor:
RICARDO ESPAÑOL CRESPO.

Gerente:
ANTONIO G. CUERPO.

Directora de producción:
MARIA LUISA SUAREZ PEREZ.

Directores de la colección:
MANUEL ALFONSECA, Doctor Ingeniero de Telecomunicación y Licenciado en Informática.
JOSE ARTECHE, Ingeniero de Telecomunicación.

Diseño y maquetación:
BRAVO-LOFISH.

Fotografía:
EQUIPO GALATA.

Dibujos:
JOSE OCHOA

TECNICAS DE PROGRAMACION: Manuel Alfonseca, Doctor Ingeniero de Telecomunicación y Licenciado en Informática. TECNICAS DE ANALISIS: José Arteche, Ingeniero en Telecomunicación. LENGUAJE MAQUINA 8086: Juan Rojas Licenciado en Ciencias Físicas e Ingeniero Industrial. PASCAL: Juan Ignacio Puyol, Ingeniero Industrial. PROGRAMAS (educativos, de utilidad, de gestión y de juegos): Francisco Morales, Técnico en Informática y colaboradores. Coordinador de AULA DE INFORMATICA APLICADA (AIA): Alejandro Marcos, Licenciado en Ciencias Químicas. BASIC: Esther Maldonado, Diplomada en Arquitectura. INFORMATICA BASICA: Virginia Muñoz, Diplomada en Informática. LENGUAJE MAQUINA Z-80: Joaquín Salvachúa, Diplomado en Telecomunicación y José Luis Tojo, Diplomado en Telecomunicación. LENGUAJE MAQUINA 6502: (desde el tomo 5): Juan José Gómez, Licenciado en Química. LOGO: Cristina Manzanera, Licenciada en Informática. APLICACIONES: Sociedad Tamariz, Diplomada en Telecomunicación. OTROS LENGUAJES (COBOL): Eloy Pérez, Licenciado en Informática. Ana Pastor, Licenciada en Informática.

Ediciones Siglo Cultural, S.A.

Dirección, redacción y administración:
Pedro Teixeira, 8, 2.ª planta. Teléf. 810 52 13. 28020 Madrid.

Publicidad:
Gofar Publicidad, S.A. Benito de Castro, 12 bis. 28028 Madrid.

Distribución en España:
COEDIS, S.A. Valencia, 245. Teléf. 215 70 97. 08007 Barcelona.
Delegación en Madrid: Serrano, 165. Teléf. 411 11 48.

Distribución en Ecuador: Muñoz Hnos.

Distribución en Perú: DISELPESA.

Distribución en Chile: Alfa Ltda.

Importador exclusivo Cono Sur:
CADE, S.R.L. Pasaje Sud América, 1532. Teléf.: 21 24 64.
Buenos Aires - 1.290. Argentina.

ISBN del tomo: 84-7688-127-4
ISBN de la obra: 84-7688-068-7

Fotocomposición:
ARTECOMP, S.A. Albarracín, 50. 28037 Madrid.

Imprime:
MATEU CROMO. Pinto (Madrid).

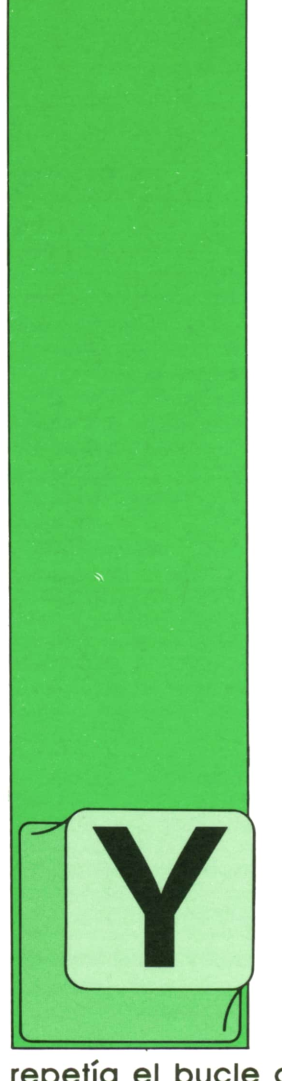
Depósito legal: M-5-677-1987
Printed in Spain - Impreso en España.

Suscripciones y números atrasados:
Ediciones Siglo Cultural, S.A.
Pedro Teixeira, 8, 2.ª planta. Teléf. 810 52 13. 28020 Madrid.

Julio, 1987.
P.V.P. Canarias: 335,–.

INDICE

BASIC



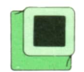


Bucles incondicionales: FOR-TO-NEXT

YA hemos estudiado la posibilidad de construir estructuras repetitivas mediante el uso combinado de las instrucciones IF-THEN y GOTO. El número de veces que se repetía el bucle dependía de la condición que marcaba el fin del mismo.

Sin embargo, el lenguage BASIC dispone de una nueva estructura que permite la realización de bucles de un modo sencillo. Esta estructura se denomina abreviadamente bucles FOR-TO-NEXT, o simplemente FOR-NEXT. Básicamente se pueden hacer las siguientes identificaciones:

* Instrucción de comienzo del bucle: FOR-TO.
* Instrucción de cierre del bucle: NEXT.



La instrucción FOR-TO

Un primer formato general para la instrucción FOR-TO es el siguiente:

FOR <variable índice>=<valor inicial>
TO <valor final>

La *variable índice* se puede asimilar al contador del bucle construido con IF-THEN, ya que su misión es la misma: contabilizar el número de veces que se ejecuta el bucle. Para ello hay que especificar un *valor inicial* y un *valor final* para dicha variable.

El número de valores diferentes que adquiere la variable índice, desde el valor inicial hasta el final, determina el número de veces que se repite el bucle. Siempre que la instrucción FOR-TO tenga el formato indicado, el incremento del valor de la variable índice será la unidad en cada ejecución del bucle.

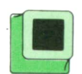
En la figura 1 podemos ver algunos ejemplos aclaratorios.

Instrucción	Valor inicial	Valor final	Valores sucesivos	N.º de ejecuciones del bucle
FOR J=1 TO 10	1	10	1,2,3,4,5,6,7,8,9,10	10
FOR K=4 TO 7	4	7	4, 5, 6, 7	4
FOR A=8 TO 9	8	9	8, 9	2



Ejemplos de instrucciones FOR-TO.

En cuanto a la variable índice, no es más que una variable numérica y, por tanto, los nombres válidos son los ya comentados en el tomo 5, a excepción del SPECTRUM, que sólo admite una letra como nombre de variable índice. Además, el AMSTRAD, el IBM y el MSX admiten variables de tipo entero.



La instrucción NEXT

El formato general es el siguiente:

NEXT <variable índice>

donce la *variable índice* especificada es la misma que figura en la instrucción FOR-TO del mismo bucle.

La instrucción NEXT tiene por objetivo incrementar el valor de la variable índice. A continuación comprueba si el nuevo valor obtenido es mayor o menor que el valor final especificado en la instrucción FOR-TO correspondiente. Si el nuevo valor es menor o igual que el valor final se repetirá la ejecución del bucle, mientras que en caso contrario el bucle no se repetirá y la ejecución del programa continuará por la instrucción siguiente el NEXT. Si no hay más instrucciones la ejecución finalizará.

Veamos un primer ejemplo. El programa 1 tiene por objeto imprimir en pantalla todos los números enteros del intervalo que deseemos, así como las sumas acumuladas correspondientes.

```
10 REM **********************
20 REM *   SUMAS ACUMULADAS   *
30 REM **********************
40 CLS
50 INPUT "NUMERO INICIAL ";NI
60 INPUT "NUMERO FINAL ";NF
70 CLS
80 IF NF<NI THEN PRINT "TECLEE PRIMERO EL MENOR":GOTO 50
90 LET SA=0
100 PRINT "NUMERO";TAB(15);"SUMA ACUMULADA"
110 PRINT "______";TAB(15);"______________"
120 PRINT :PRINT
130 FOR I=NI TO NF
140 LET SA=SA+I
150 PRINT I;TAB(15);SA
160 NEXT I
```

En la línea 130 se inicia el bucle, especificando los valores inicial y final para la variable índice I. En este caso ambos valores son variables, ya que dependen de los números inicial y final del intervalo que deseamos sumar. De todas formas dichos valores pueden ser también constantes numéricas o expresiones aritméticas.

Cada uno de los valores que toma la variable I se suma a la variable SA en la línea 140, de modo que SA actúa de acumulador.

La línea 150 se encarga de imprimir en pantalla en cada ejecución del bucle el valor actual de I, así como la suma acumulada en ese momento.

Finalmente, en la línea 160 se encuentra la instrucción NEXT que cierra el bucle.

En la figura 2 podemos ver el aspecto de la pantalla, tras una posible ejecución de este programa.

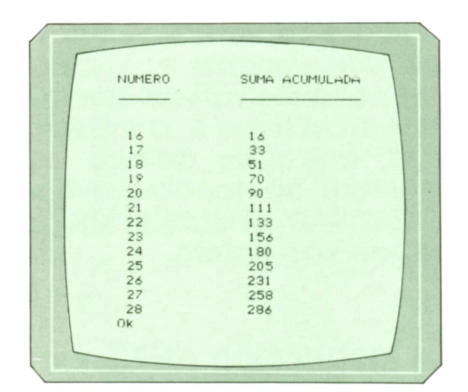
Es importante tener en cuenta que el valor de la variable índice a la salida de un bucle FOR-TO-NEXT será siempre igual

al valor final especificado más el incremento propio del bucle.

El programa 2 es un ejemplo típico de aplicación de los bucles FOR-TO-NEXT. Su objetivo es calcular el factorial de un número cualquiera. Recordemos que la expresión matemática para el cálculo del factorial es la siguiente:

FACTORIAL DE N=N!=1*2*3*...*(N-2)*(N-1)*N

Por tanto, el factorial de un número N es igual al producto de todos los números naturales comprendidos entre 1 y N.





Presentación en pantalla del programa 1.

```
10 REM ****************************
20 REM *  FACTORIAL DE UN NUMERO  *
30 REM ****************************
40 CLS
50 INPUT "¿ DE QUE NUMERO DESEA CALCULAR
EL FACTORIAL ? ";N
60 CLS
70 IF N>33 THEN PRINT "NO PUEDO CALCULAR
FACTORIALES DE NUMEROS MAYORES DE 33": GOTO 50
80 LET FAC=1
90 FOR I=2 TO N
100 LET FAC=FAC*I
110 NEXT I
120 PRINT "NUMERO    =",N
130 PRINT :PRINT
140 PRINT "FACTORIAL =",FAC
```

La condición de la línea 70 evita que el ordenador trate de calcular factoriales de números mayores que 33, ya que no tiene capacidad para manejar números tan grandes.

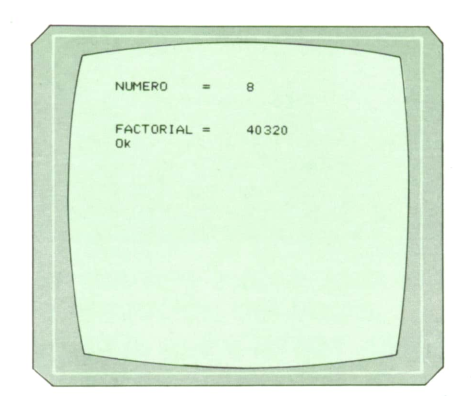
En la línea 80 se establece la variable FAC que va a ser en la que se acumulen posteriormente todas las multiplicaciones. Se la asigna el valor 1, puesto que es el elemento neutro de la multiplicación, además de ser el primer número a multiplicar.

A continuación se inicia el bucle en la línea 90. La variable índice I va a tomar todos los valores entre 2 y N, que son los que hay que multiplicar.

En la línea 100 se efectúan todas las multiplicaciones, una por cada ejecución del bucle.

La línea 110 marca el final del bucle. Finalmente se imprime en pantalla el resultado tal y como muestra la figura 3.

El programa 3 tiene como objetivo localizar el mayor y el menor de una serie de datos. En este caso concreto los datos son las edades de un conjunto de personas.





Presentación en pantalla del programa 2.

```
10 REM *****************
20 REM *  MAYOR Y MENOR *
30 REM *****************
40 CLS
```

```
50 LET MA=0:LET ME=100
60 LET A$="":LET E$=""
70 INPUT "NUMERO DE DATOS ";D
80 IF D<1 THEN GOTO 70
90 CLS
100 FOR I=1 TO D
110 INPUT "NOMBRE ";N$
120 INPUT "EDAD ";E
130 IF E<=0 OR E>100 THEN GOTO 120
140 IF E>MA THEN LET MA=E:LET A$=N$
150 IF E<ME THEN LET ME=E:LET E$=N$
160 CLS
170 NEXT I
180 PRINT "DATOS DEL MAYOR"
190 PRINT "_______________"
200 PRINT
210 PRINT "NOMBRE:",A$
220 PRINT "EDAD   :",MA
230 PRINT :PRINT
240 PRINT "DATOS DEL MENOR"
250 PRINT "_______________"
260 PRINT
270 PRINT "NOMBRE:",E$
280 PRINT "EDAD   :",ME
```

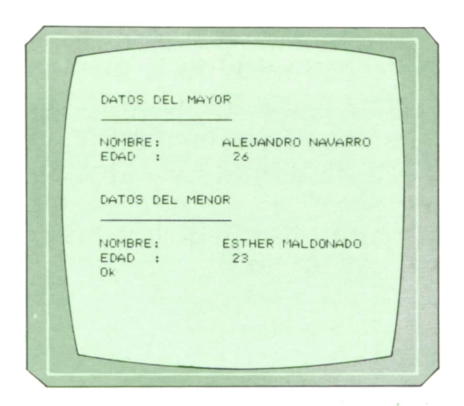
En la línea 50 establecemos unas edades ficticias para comenzar: 0 para el mayor y 100 para el menor. Evidentemente, cualquier edad que tecleemos posteriormente será mayor que 0 y menor que 100, por tanto, estos valores de MA y ME cambiarán durante la ejecución. Del mismo modo, en la línea 60 asignamos la cadena vacía ("") a dos variables A$ y E$ que almacenarán posteriormente los nombres del mayor y el menor, respectivamente.

En este programa es necesario conocer de antemano el número de datos que se va a introducir, puesto que, de lo contrario, no podríamos utilizar un bucle FOR-TO-NEXT.

La línea 100 marca el inicio del bucle, que va a actuar como mero contador de los datos que se van a introducir en las líneas 110 y 120.

La condición de la línea 140 se encarga de comprobar si la edad tecleada es mayor que la almacenada en MA, en cuyo caso la nueva edad pasará a almacenarse en MA, desapareciendo la antigua, y el nombre de la persona correspondiente pasará a almacenarse en A$. La condición de la línea 150 es análoga a la anterior, pero para localizar el menor.

Cuando la ejecución sale del bucle las edades del mayor y el menor estarán en MA y ME, respectivamente, así como los nombres correspondientes en A$ y E$, por tanto, se imprimirán en pantalla los resultados tal y como muestra la figura 4.



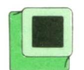


Presentación en pantalla del programa 3.

La mayoría de los ordenadores permiten no indicar la variable índice correspondiente en la instrucción NEXT, por tanto, la siguiente estructura sería correcta:

```
FOR I=1 TO 20
.
.
.
NEXT
```

De los ordenadores en estudio, únicamente es necesario indicar la variable índice del NEXT en el SPECTRUM.



El incremento: STEP

Hasta ahora los bucles FOR-TO-NEXT que hemos visto tenían siempre un incremento de la variable índice de la unidad. Sin embargo, podemos establecer un incremento distinto mediante la instrucción STEP que se utiliza ligada a la instrucción FOR-TO con el siguiente formato:

FOR <variable índice>=<valor inicial> TO <valor final> STEP <incremento>

Evidentemente, el incremento es de naturaleza numérica, al igual que los valores inicial y final, por tanto, puede ser una constante, una variable o una expresión artimética. Asimismo puede ser positivo, negativo, entero o decimal.

El programa 4 tiene por objeto imprimir en pantalla la progresión aritmética que deseemos en el intervalo que queramos. Recordemos que una progresión aritmética es una serie de números tales que un número cualquiera N de la progresión es igual al que le precede, N-1, más una razón de la progresión, R. Por ejemplo:

9, 12, 15, 18, 21, 24

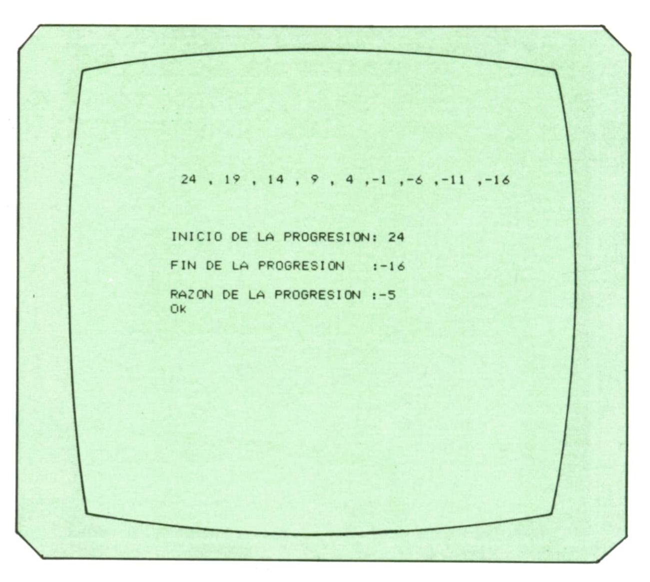
es una progresión aritmética de razón R = 3.

```
10 REM ******************************
20 REM *  PROGRESIONES ARITMETICAS  *
30 REM ******************************
40 CLS
50 INPUT "NUMERO INICIAL ";NI
60 INPUT "NUMERO FINAL ";NF
70 INPUT "RAZON ";R
80 CLS
90 IF R>0 AND NI>NF THEN GOTO 50
100 IF R<0 AND NI<NF THEN GOTO 50
110 FOR I=NI TO NF STEP R
120 PRINT I;",";
130 NEXT I
140 PRINT :PRINT:PRINT :PRINT
150 PRINT "INICIO DE LA PROGRESION:";NI
160 PRINT
170 PRINT "FIN DE LA PROGRESION    :";NF
180 PRINT
190 PRINT "RAZON DE LA PROGRESION :";R
```

Las condiciones de las líneas 90 y 100 impiden que tecleemos un incremento positivo cuando el número inicial es mayor que el final, y viceversa.

El bucle FOR-TO-NEXT de las líneas 110 a 130 se encarga de imprimir en pantalla la progresión.

En la figura 5 podemos ver el aspecto de la pantalla, tras una posible ejecución.





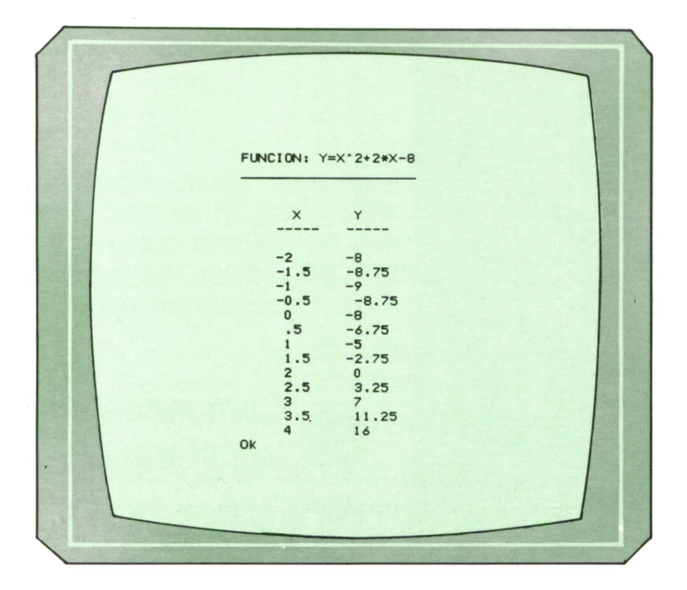
Presentación en pantalla del programa 4.

Finalmente, el programa 5 permite determinar los valores de la función $y = x^2 + 2x + 8$ para distintos valores de x con el incremento deseado. Cuanto más pequeño sea el incremento los valores obtenidos serán más próximos, con lo cual podríamos hacer posteriormente una representación gráfica muy exacta.

```
10 REM *************************************
20 REM *  TABLA DE VALORES DE UNA FUNCION  *
30 REM *************************************
40 CLS
50 INPUT "VALOR INICIAL ";VI
60 INPUT "VALOR FINAL ";VF
70 INPUT "INCREMENTO ";I
80 IF I>0 AND VI>VF THEN GOTO 50
90 IF I<0 AND VI<VF THEN GOTO 50
100 CLS
110 PRINT "FUNCION: Y=X^2+2*X-8"
120 PRINT "______________________"
130 PRINT :PRINT
140 PRINT TAB(7);"X";TAB(14);"Y"
150 PRINT TAB(5);"-----";TAB(13);"-----"
160 PRINT
170 FOR X=VI TO VF STEP I
180 LET Y=X^2+2*X-8
190 PRINT TAB(5);X;TAB(13);Y
200 NEXT X
```

La figura 6 muestra la presentación en pantalla, tras una posible ejecución.

Tras estos ejemplos podemos sacar algunas conclusiones. Por ejemplo, si el valor inicial del bucle es menor que el final el incremento debe ser siempre positivo, mientras que si el valor inicial es mayor que el final el incremento tiene que ser negativo. Si no cumplimos estos requisitos habremos creado un bucle infinito.

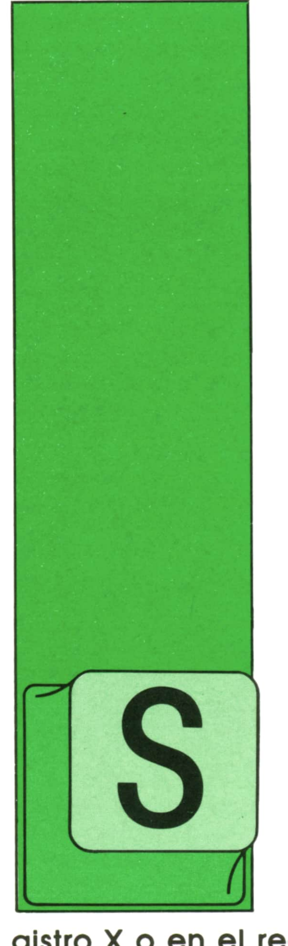




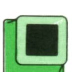
Presentación en pantalla del programa 5.

MAQUINA 6502

COMMODORE 64



Comandos de transferencia



SON aquellos que permiten copiar el contenido de un registro a otro dentro del microprocesador, de tal manera que se puede cargar al ACU lo que haya en el registro X o en el registro Y. Esto es importante, ya que algunos comandos trabajan sólo con el ACU. El contenido del registro origen permanece invariable.

Sólo hay una excepción; no se pueden transferir los registros X e Y entre sí.

Todos ellos son de 1 byte y por tanto, no necesitan operandos.

```
TAX    X = A
TXA    A = X
TAY    Y = A
TYA    A = Y
TSX    X = S. Pointer
TXS    S. Pointer = X
```

El registro Stack Pointer (apuntador a pila) sólo se puede transferir al registro X, y su utilización se verá más adelante.

Con los cuatro primeros comandos se alteran los flags Zero y Negative del registro de estado, dependiendo del valor transferido.

Con los dos últimos, en los que interviene el Stack Pointer, no se altera ningún flag, ya que el SP no es un registro normal de trabajo del procesador.

Por último, veamos los códigos correspondientes a estos comandos.

Comando	Código
TAX	$AA
TXA	$8A
TAY	$A8
TYA	$98
TSX	$BA
TXS	$9A

Comandos aritméticos



Son comandos que se utilizan para hacer adiciones y sustracciones con los registros del procesador.

Constan de dos operandos, el primero debe estar en el ACU, y el segundo se extrae de la memoria. El resultado se lleva siempre al ACU.

Así, por ejemplo, el comando

```
ADC # $A0
```

podríamos "traducirlo" como: Sumar al contenido del ACU, el número $A0, y el resultado ponerlo en el ACU.

Igual que con los comandos de carga y almacenamiento existen varios modos de direccionar estos comandos que están incluidos al final de los comandos aritméticos.

Ahora vamos a abordar un problema que todavía no nos había surgido.

Al sumar dos números de 8 bits (0-255), el resultado podría ser mayor que 255, cifra que no podría representarse en un solo registro de 8 bits. Ha ocurrido un desbordamiento, palabra que fue introducida al hablar de los registros propios del microprocesador.

Vamos a ver qué es lo que ocurre en estos casos: Supongamos que el ACU contiene el número $A3 = 163 = = %10100011, y a continuación introducimos el comando a) ADC # $15; b) ADC # $98.

a) La adición sería:

```
  $A3 = %  10100011
+ $15 = %  00010101
-------------------
  $B8 = %  10111000
```

Como puede observarse la suma binaria presenta cuatro posibilidades:

0 + 0 = 0
0 + 1 = 1
1 + 0 = 1
1 + 1 = 0 (más el desbordamiento o "llevarse una")

Bien, en este caso el resultado de la adición es menor de 255 = % 11111111 = $FF y no ocurre nada anormal, pues el número $B8 puede ser representado perfectamente con 8 bits en el ACU.

b) La adición sería:

```
  $A3 = %  10100011
+ $98 = %  10011000
-------------------
  $13B=   100111011
```

Se ha producido un desbordamiento que excede los 8 bits. Es ahora cuando se activa el Carry flag del registro de estado: (C=1). Es por ello, que a veces se le denomina "noveno bit del ACU".

Según lo que acabamos de ver, se deduce que el Carry flag debe ser desactivado (C=0) antes de cada operación de adición, pues si no, podría aparecer un desbordamiento después de operar, que en realidad no existe.

Además del Carry flag se podrán activar los flags Zero y Negative dependiendo del resultado de la adición. La sustracción aritmética se realiza de forma análoga a la adición. En este caso se resta del ACU el byte direccionado. Al igual que en la adición puede ocurrir un desbordamiento pero negativo, es decir, un número menor que cero.

Cuando se produce un desbordamiento negativo el Carry flag se desactiva. Por tanto, antes de realizar una substracción, se debe activar el Carry flag, para evitar la aparición de un desbordamiento que en realidad no ha tenido lugar si antes de la operación ya estaba desactivado dicho flag.

La resta binaria, al igual que la suma, presenta cuatro posibilidades:

0 − 0 = 0
0 − 1 = 1 (más el desbordamiento negativo)
1 − 0 = 1
1 − 1 = 0

Supongamos que hemos cargado el ACU con el número $0B = %00001011 = 11, y a continuación introducimos el comando SBC $20 = 00100000 = 32.

La representación sería la siguiente:

```
  %  00001011
− %  00100000
-------------
  %  11101011
```

Se obtiene el número $EB=235, produciéndose un desbordamiento negativo, y desactivándose, por tanto, el Carry flag.

Veamos cómo se interpreta un resultado de este tipo.

Si hubiésemos hecho la resta al revés, el resultado hubiera sido % 00010101 = $15 = 21. Como puede observarse la suma de los dos resultados es 256, que como usted ya sabe son el número de combinaciones posibles dentro de un registro de 8 bits.

Pues bien, se pueden hacer dos cosas:

— Hacer la resta normalmente, invertir todos los bits, y sumar uno al resultado.

%11101011 → %00010101 = 21 = $15

— Hacer la resta al revés, es decir, el byte direccionado menos el número que se halle en el ACU.

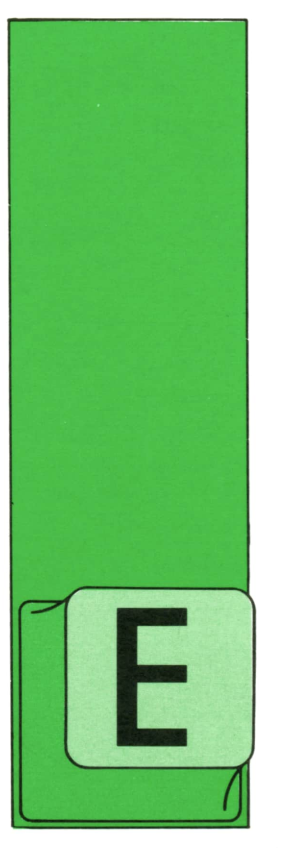
%00100000 − %00001011 = %00010101

Para terminar con los comandos aritmético, se presenta aquí una lista de sus códigos en sus diferentes modos de direccionamiento.

Modos de direccionamiento	ADC	SBC
Inmediato	$69	$E9
Absoluto	$6D	$FD
Zero page	$65	$E5
Absoluto indexado por X	$7D	$FD
Absoluto indexado por Y	$79	$F9
Zeropage indexado por X	$75	$F5
Indirecto indexado por Y	$71	$F1
Indirecto indexado por X	$61	$E1

PROGRAMAS

EDUCATIVOS • DE UTILIDAD • DE GESTION • DE JUEGOS





Programa: Juego de Damas

EL juego de las DAMAS, junto con el AJEDREZ, EL PARCHIS y LA OCA, es uno de los más jugados y más apreciados en España. A continuación proponemos un programa que nos permitirá jugarlo de forma que nuestro oponente sea el ordenador.

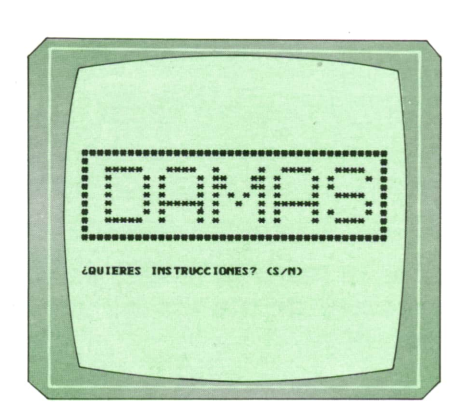




Fig. 1.

Poco hay que explicar sobre su funcionamiento, ya que es conocido por todos. Lo único que hay que decir sobre el programa es explicar cómo funciona.

Las fichas del ordenador estarán marcadas con una X, y las del jugador, con una O. Estas letras estarán en minúscula al empezar el juego. Cuando cualquiera de los dos jugadores haga DAMA, entonces la ficha con la que haya llegado al otro extremo del tablero se pondrá en mayúscula.

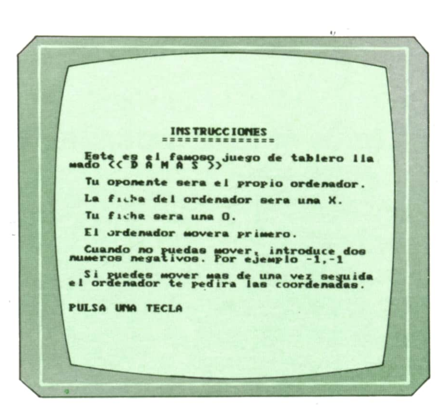




Fig. 2.

El tablero estará representado tal y como aparece en la figura 3. Para realizar un movimiento tendrás que dar las coordenadas de origen, pulsa ENTER y a continuación las de destino. Primero tendrás que dar la coordenada en X y después la coordenada en Y. Si al realizar un movimiento te comes una de las fichas del oponente, el ordenador te preguntará si quieres realizar otro movimiento para comerte otra ficha. En caso negativo tienes que contestar con dos números negativos.

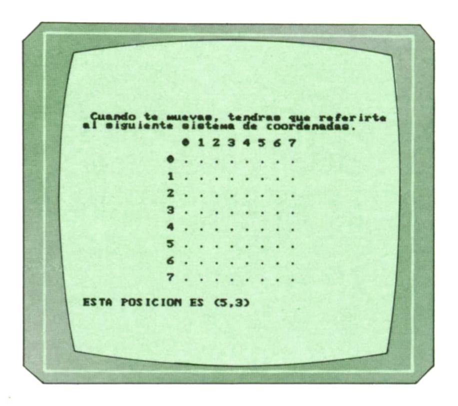




Fig. 3.

```
100 REM ******************************************
110 REM *                                        *
120 REM *   DDDDD   AAAA  M     M  AAAA   SSSS   *
130 REM *   D    D A    A MM   MM A    A S    S  *
140 REM *   D    D A    A MM   MM A    A S       *
150 REM *   D    D A    A M M M M A    A  SSSS   *
160 REM *   D    D AAAAAA M  M  M AAAAAA      S  *
170 REM *   D    D A    A M     M A    A S    S  *
180 REM *   DDDDD  A    A M     M A    A  SSSS   *
190 REM *                                        *
200 REM ******************************************
210 REM
220 REM ******************************************
230 REM ********* (c) Ed. Siglo Cultural *********
240 REM ********* (c) 1987               *********
250 REM ******************************************
260 REM
270 CLS
280 LOCATE 12,9
290 PRINT "EDICIONES SIGLO CULTURAL"
300 LOCATE 14,14
310 PRINT "PRESENTA ..."
320 FOR I=1 TO 1000
330 NEXT I
340 LOCATE 23,1
350 PRINT "########################################"
360 PRINT "#                                      #"
370 PRINT "#  *****   ****  *     *  ****   ****  #"
380 PRINT "#  *    * *    * **   ** *    * *    * #"
390 PRINT "#  *    * *    * **   ** *    * *      #"
400 PRINT "#  *    * *    * * * * * *    *  ****  #"
410 PRINT "#  *    * ****** *  *  * ******      * #"
420 PRINT "#  *    * *    * *     * *    * *    * #"
430 PRINT "#  *****  *    * *     * *    *  ****  #"
440 PRINT "#                                      #"
450 PRINT "########################################"
460 FOR I=1 TO 12
470    PRINT
480 NEXT I
490 FOR I=1 TO 1000
500 NEXT I
510 LOCATE 15,1
520 PRINT "(QUIERES INSTRUCCIONES? (S/N)"
530 LET A$=INKEY$
540 IF A$="S" OR A$="s" THEN GOSUB 2340:GOTO 570
550 IF A$="N" OR A$="n" THEN GOTO 570
560 GOTO 530
570 CLS
580 PRINT TAB(16);"D A M A S"
590 PRINT TAB(15);"-----------"
600 FOR I=1 TO 8
610    LOCATE 3+2*I,14
620    FOR J=1 TO 8
630       PRINT ". ";
640    NEXT J
650 NEXT I
660 FOR I=0 TO 7
670    LOCATE 3,13+2*I
680    PRINT I
690 NEXT I
700 FOR I=0 TO 7
710    LOCATE 5+2*I,11
720    PRINT I
730 NEXT I
740 REM
750 REM ************************************
760 REM **** INICIALIZACION DEL PROGRAMA ****
770 REM ************************************
```

```
780 REM
790 DIM R(4)
800 DIM S(7,7)
810 LET G=-1
820 LET R(0)=-99
830 DATA 1,0,1,0,0,0,-1,0,0,1,0,0,0,-1,0,-1,15
840 FOR X=0 TO 7
850    FOR Y=0 TO 7
860       READ J
870       IF J=15 THEN GOTO 900
880       S(X,Y)=J
890       GOTO 920
900       RESTORE
910       READ S(X,Y)
920    NEXT Y
930 NEXT X
940 REM
950 REM **********************
960 REM * PROGRAMA PRINCIPAL *
970 REM **********************
980 REM
990 FOR X=0 TO 7
1000    FOR Y=0 TO 7
1010       IF S(X,Y)>-1 THEN GOTO 1040
1020       IF S(X,Y)=-1 THEN FOR A=-1 TO 1 STEP 2:B=G:GOSUB 1070:NEXT A
1030       IF S(X,Y)=-2 THEN FOR A=-1 TO 1 STEP 2:FOR B=-1 TO 1 STEP 2:GOSUB 107
0:NEXT B:NEXT A
1040    NEXT Y
1050 NEXT X
1060 GOTO 1300
1070 LET U=X+A
1080 LET V=Y+B
1090 IF U<0 OR V<0 OR U>7 OR V>7 THEN GOTO 1160
1100 IF S(U,V)=0 THEN GOSUB 1170:GOTO 1160
1110 IF S(U,V)<0 THEN GOTO 1160
1120 LET U=U+A
1130 LET V=V+B
1140 IF U<0 OR V<0 OR U>7 OR V>7 THEN GOTO 1160
1150 IF S(U,V)=0 THEN GOSUB 1170
1160 RETURN
1170 IF V=0 AND S(X,Y)=-1 THEN LET Q=Q+2
1180 IF ABS(Y-V)=2 THEN LET Q=Q+5
1190 IF Y=7 THEN LET Q=Q-2
1200 IF U=0 OR U=7 THEN LET Q=Q+1
1210 FOR C=-1 TO 1 STEP 2
1220    IF U+C<0 OR U+C>7 OR V+G<0 THEN GOTO 1260
1230    IF S(U+C,V+G)<0 THEN LET Q=Q+1:GOTO 1260
1240    IF U-C<0 OR U-C>7 OR V-G>7 THEN GOTO 1260
1250    IF S(U+C,V+G)>0 AND (S(U-C,V-G)=0 OR (U-C=X AND V-G=Y)) THEN LET Q=Q-2
1260 NEXT C
1270 IF Q>R(0) THEN LET R(0)=Q:LET R(1)=X:LET R(2)=Y:LET R(3)=U:LET R(4)=V
1280 LET Q=0
1290 RETURN
1300 IF R(0)=-99 THEN GOTO 2240
1310 LOCATE 21,1
1320 PRINT SPACE$(40)
1330 LOCATE 21,1
1340 PRINT "ME MUEVO DESDE";R(1);",";R(2);"A";R(3);",";R(4);
1350 LET R(0)=-99
1360 IF R(4)=0 THEN LET S(R(3),R(4))=-2:GOTO 1520
1370 LET S(R(3),R(4))=S(R(1),R(2))
1380 LET S(R(1),R(2))=0
1390 IF ABS(R(1)-R(3))<>2 THEN GOTO 1520
1400 LET S((R(1)+R(3))/2,(R(2)+R(4))/2)=0
1410 LET X=R(3)
1420 LET Y=R(4)
1430 IF S(X,Y)=-1 THEN LET B=-2:FOR A=-2 TO 2 STEP 4:GOSUB 1470:NEXT A
1440 IF S(X,Y)=-2 THEN FOR A=-2 TO 2 STEP 4:FOR B=-2 TO 2 STEP 4:GOSUB 1470:NEXT
```

```
  B:NEXT A
1450 IF R(0)<>-99 THEN PRINT "Y A" R(3);",";R(4);:R(0)=-99:GOTO 1360
1460 GOTO 1520
1470 LET U=X+A
1480 LET V=Y+B
1490 IF U<0 OR U>7 OR V<0 OR V>7 THEN GOTO 1510
1500 IF S(U,V)=0 AND S(X+A/2,Y+B/2)>0 THEN GOSUB 1170
1510 RETURN
1520 REM
1530 REM **************************
1540 REM *** DIBUJO DEL TABLERO ***
1550 REM **************************
1560 REM
1570 FOR Y=0 TO 7
1580    LOCATE 5+2*Y,14
1590    FOR X=0 TO 7
1600        IF S(X,Y)=0 THEN PRINT ". ";
1610        IF S(X,Y)=1 THEN PRINT "o ";
1620        IF S(X,Y)=-1 THEN PRINT "x ";
1630        IF S(X,Y)=-2 THEN PRINT "X ";
1640        IF S(X,Y)=2 THEN PRINT "O ";
1650    NEXT X
1660 NEXT Y
1670 PRINT
1680 FOR L=0 TO 7
1690    FOR M=0 TO 7
1700        IF S(L,M)=1 OR S(L,M)=2 THEN LET Z=1
1710        IF S(L,M)=-1 OR S(L,M)=-2 THEN LET T=1
1720    NEXT M
1730 NEXT L
1740 REM
1750 REM **************************
1760 REM * MOVIMIENTO DEL JUGADOR *
1770 REM **************************
1780 REM
1790 IF Z<>1 THEN GOTO 2240
1800 IF T<>1 THEN GOTO 2290
1810 LET T=0
1820 LET Z=0
1830 LOCATE 21,1
1840 PRINT SPACE$(40)
1850 LOCATE 21,1
1860 INPUT "MUEVES DESDE";E,H
1870 LET X=E
1880 LET Y=H
1890 IF S(X,Y)<=0 THEN GOTO 1830
1900 LOCATE 21,1
1910 PRINT SPACE$(40)
1920 LOCATE 21,1
1930 INPUT "HASTA ";A,B
1940 LET X=A
1950 LET Y=B
1960 IF S(X,Y)=0 AND ABS(A-E)<=2 AND ABS(A-E)=ABS(B-H) THEN GOTO 2050
1970 LOCATE 21,1
1980 PRINT SPACE$(40)
1990 LOCATE 21,1
2000 PRINT "MOVIMIENTO INVALIDO"
2010 BEEP
2020 FOR Z=1 TO 1000
2030 NEXT Z
2040 GOTO 1900
2050 LET I=46
2060 LET S(A,B)=S(E,H)
2070 LET S(E,H)=0
2080 IF ABS(E-A)<>2 THEN GOTO 2220
2090 LET S((E+A)/2,(H+B)/2)=0
2100 LOCATE 21,1
2110 PRINT SPACE$(40)
```

```
2120 LOCATE 21,1
2130 INPUT "Y HASTA ";A1,B1
2140 IF A1<0 THEN GOTO 2220
2150 IF S(A1,B1)<>0 OR ABS(A1-A)<>2 OR ABS(B1-B)<>2 THEN GOTO 2100
2160 LET E=A
2170 LET H=B
2180 LET A=A1
2190 LET B=B1
2200 LET I=I+15
2210 GOTO 2060
2220 IF B=7 THEN LET S(A,B)=2
2230 GOTO 990
2240 LOCATE 21,1
2250 PRINT SPACE$(40)
2260 LOCATE 21,1
2270 PRINT "TE HE GANADO. ERES MALISIMO"
2280 GOTO 2920
2290 LOCATE 21,1
2300 PRINT SPACE$(40)
2310 LOCATE 21,1
2320 PRINT "ME HAS GANADO. DE CHORRA."
2330 GOTO 2920
2340 CLS
2350 PRINT TAB(14);"INSTRUCCIONES"
2360 PRINT TAB(13);"==============="
2370 PRINT
2380 PRINT "  Este es el famoso juego de tablero lla"
2390 PRINT "mado << D A M A S >>"
2400 PRINT
2410 PRINT "  Tu oponente sera el propio ordenador."
2420 PRINT
2430 PRINT "  La ficha del ordenador sera una X."
2440 PRINT
2450 PRINT "  Tu ficha sera una O.
2460 PRINT
2470 PRINT "  El ordenador movera primero."
2480 PRINT
2490 PRINT "  Cuando no puedas mover, introduce dos "
2500 PRINT "números negativos. Por ejemplo -1,-1"
2510 PRINT
2520 PRINT "  Si puedes mover más de una vez seguida"
2530 PRINT "el ordenador te pedirá las coordenadas. "
2540 LOCATE 22,1
2550 PRINT "PULSA UNA TECLA"
2560 LET A$=INKEY$
2570 IF A$="" THEN GOTO 2560
2580 CLS
2590 PRINT " Cuando te muevas, tendrás que referirte"
2600 PRINT "al siguiente sistema de coordenadas."
2610 FOR I=1 TO 8
2620    LOCATE 4+2*I,14
2630    FOR J=1 TO 8
2640       PRINT ". ";
2650    NEXT J
2660 NEXT I
2670 FOR I=0 TO 7
2680    LOCATE 4,13+2*I
2690    PRINT I
2700 NEXT I
2710 FOR I=0 TO 7
2720    LOCATE 6+2*I,11
2730    PRINT I
2740 NEXT I
2750 LOCATE 23,1
2760 PRINT "ESTA POSICION ES (5,3)";
2770 FOR I=1 TO 10
2780    LOCATE 12,24
2790    PRINT " "
```

```
2800     FOR J=1 TO 200
2810     NEXT J
2820     LOCATE 12,24
2830     PRINT "."
2840     FOR J=1 TO 200
2850     NEXT J
2860 NEXT I
2870 LOCATE 22,1
2880 PRINT "  Primero tienes que dar la coordenada  en X y después la coordenada
 en Y.";
2890 FOR I=1 TO 1000
2900 NEXT I
2910 RETURN
2920 REM
2930 REM ***********************
2940 REM * OTRA PARTIDA? (S/N) *
2950 REM ***********************
2960 REM
2970 FOR I=1 TO 2000
2980 NEXT I
2990 LOCATE 21,1
3000 PRINT SPACE$(40)
3010 LOCATE 21,1
3020 PRINT "OTRA PARTIDA? (S/N)"
3030 LET A$=INKEY$
3040 IF A$="" THEN GOTO 3030
3050 IF A$="S" OR A$="s" THEN RUN
3060 IF A$="N" OR A$="n" THEN GOTO 3080
3070 GOTO 3030
3080 REM
3090 REM *************
3100 REM * A D I O S *
3110 REM *************
3120 REM
3130 CLS
3140 PRINT "ADIOS. ME HA ENCANTADO JUGAR CONTIGO."
3150 PRINT:PRINT:PRINT
3160 END
```

El programa está realizado en un IBM bajo GWBASIC. Para utilizar este mismo programa en el COMMODORE, AMSTRAD y MSX hay que realizar los siguientes cambios:

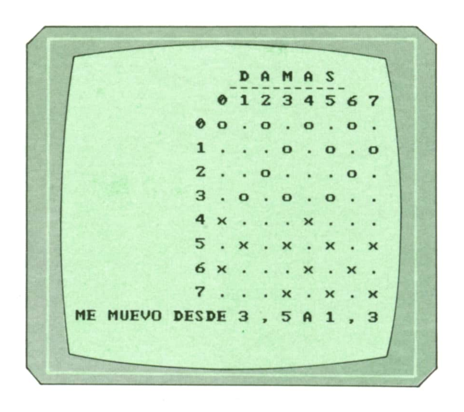
COMMODORE

```
270 PRINT CHR$(147)
280 POKE 214,11:POKE 211,8
300 POKE 214,13:POKE 211,13
340 POKE 214,23:POKE 211,0
510 POKE 214,20:POKE 211,0
530 GET A$
570 PRINT CHR$(147)
610 POKE 214,2+2*I:POKE 211,13
670 POKE 214,2:POKE 211,12+2*I
710 POKE 214,4+2*I:POKE 211,10
1310 POKE 214,21:POKE 211,0
1320 PRINT SPC(40)
1330 POKE 214,21:POKE 211,0
1580 POKE 214,4+2*Y:POKE 211,13
1830 POKE 214,21:POKE 211,0
1840 PRINT SPC(40)
1850 POKE 214,21:POKE 211,0
1900 POKE 214,21:POKE 211,0
1910 PRINT SPC(40)
1920 POKE 214,21:POKE 211,0
1970 POKE 214,21:POKE 211,0
1980 PRINT SPC(40)
1990 POKE 214,21:POKE 211,0
2010 REM
2100 POKE 214,21:POKE 211,0
2110 PRINT SPC(40)
```

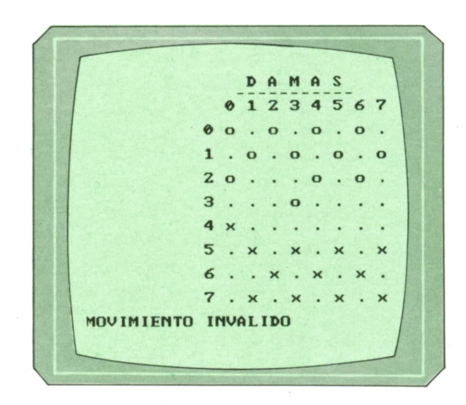
```
2120 POKE 214,21:POKE 211,0
2240 POKE 214,21:POKE 211,0
2250 PRINT SPC(40)
2260 POKE 214,21:POKE 211,0
2290 POKE 214,21:POKE 211,0
2300 PRINT SPC(40)
2310 POKE 214,21:POKE 211,0
2340 PRINT CHR$(147)
2540 POKE 214,22:POKE 211,0
2560 GET A$
2580 PRINT CHR$(147)
2620 POKE 214,3+2*I:POKE 211,13
2680 POKE 214,3:POKE 211,12+2*I
2720 POKE 214,5+2*I:POKE 211,10
2750 POKE 214,23:POKE 211,0
2780 POKE 214,11:POKE 211,23
2820 POKE 214,11:POKE 211,23
2870 POKE 214,22:POKE 211,0
2990 POKE 214,21:POKE 211,0
3000 PRINT SPC(40)
3010 POKE 214,21:POKE 211,0
3030 GET A$
3130 PRINT CHR$(147)
```

AMSTRAD Y MSX

```
280 LOCATE 9,12
340 LOCATE 1,23
510 LOCATE 1,15
610 LOCATE 14,3+2*I
670 LOCATE 13+2*I,3
710 LOCATE 11,5+2*I
1310 LOCATE 1,21
1330 LOCATE 1,21
1580 LOCATE 14,5+2*Y
1830 LOCATE 1,21
1850 LOCATE 1,21
1900 LOCATE 1,21
1920 LOCATE 1,21
1970 LOCATE 1,21
1990 LOCATE 1,21
2100 LOCATE 1,21
2120 LOCATE 1,21
2240 LOCATE 1,21
2260 LOCATE 1,21
2290 LOCATE 1,21
2310 LOCATE 1,21
2540 LOCATE 1,22
2620 LOCATE 14,4+2*I
2720 LOCATE 11,6+2*I
2750 LOCATE 1,23
2780 LOCATE 24,12
2820 LOCATE 24,12
2870 LOCATE 1,22
2990 LOCATE 1,21
3010 LOCATE 1,21
```




El programa, en plena ejecución.




El programa testea si nuestros movimientos son válidos.

Programa: Comecocos para Spectrum

El siguiente programa nos va a permitir jugar al famoso COMECOCOS. Poco hay que decir sobre el objetivo del juego, ya que es conocido por todo el mundo. Todo consiste en mover un pequeño personaje en forma de queso por el laberinto que hay en la pantalla, y comerse todos los puntos antes de que los fantasmas del laberinto te coman a ti.

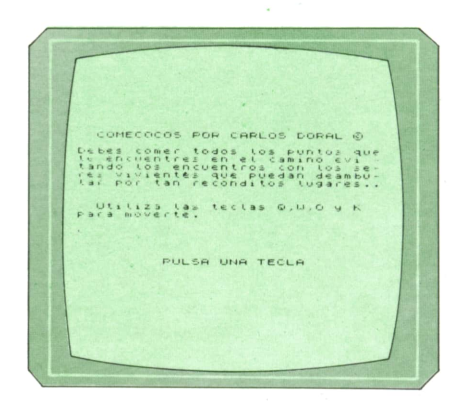
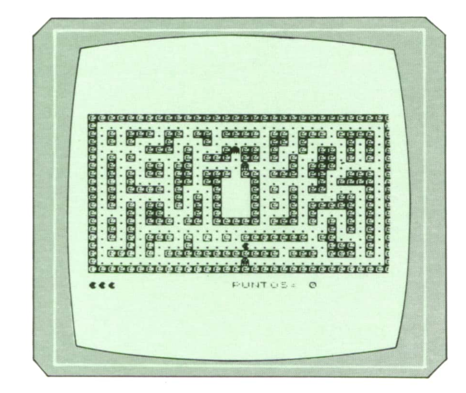



Fig. 6.

Para mover el quesito por la pantalla tienes que utilizar las siguientes teclas:

Q — Izquierda
W — Derecha
O — Arriba
K — Abajo




El laberinto donde está perdido el quesito.

```
 10 REM ***********************
 20 REM ****** COMECOCOS ******
 30 REM ***********************
 40 REM *** POR CARLOS DORAL **
 50 REM ***********************
 55 REM
 60 BORDER 0
 70 PAPER 0
 80 INK 7
 90 CLS
100 LET a$="  COMECOCOS POR CARLOS DORAL"
110 FOR f=LEN a$ TO 3 STEP -1
120    FOR c=f-3 TO f-2
130       PRINT AT 5,c;" "; INK 2; PAPER 5;a$(f TO f)
140    NEXT c
150 NEXT f
160 PRINT
170 PRINT "Debes comer todos los puntos quete encuentres en el camino evi -tand
o los encuentros con los se-res vivientes que puedan deambu-lar por tan recondit
os lugares.."
172 PRINT : PRINT
174 PRINT "  Utiliza las teclas Q,W,O y K  para moverte."
180 PRINT AT 21,9; FLASH 1;"PULSA UNA TECLA"
190 IF INKEY$="" THEN  GO TO 190
200 CLS
210 LET vi=3
220 LET v$=CHR$ 144+CHR$ 144+CHR$ 144
230 LET pu=0
240 LET p=0
250 GO SUB 1870
255 REM
260 REM **********************
270 REM *DIBUJO DEL LABERINTO*
280 REM **********************
290 REM
310 PRINT "@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@"
320 PRINT "@..............................@"
330 PRINT "@.@.@@@.@@.@@.@@@@.@@.@@@.@@@@.@"
340 PRINT "@.@.@...@...@......@@.@....@.@.@"
350 PRINT "@.....@@@.@...@.@@.@..@.@@.@.@.@"
360 PRINT "@.@.@.....@.@@@.@@.@.@..@@.@.@.@"
370 PRINT "@.@...@@.@@..........@.@@@.....@"
380 PRINT "@.@.@.@@.@@.@@@ @@.@@@.@@..@@@.@"
390 PRINT "@.@.@....@@.@@   @......@.@@@@.@"
400 PRINT "@.@.@@@@.@...@   @.@.@.@@....@.@"
410 PRINT "@.@......@.@.@   @...@....@@.@.@"
420 PRINT "@.@.@.@@.@.@.@   @.@.@.@@@@@.@.@"
430 PRINT "@...@.@@...@.@   @...@.@...@.@.@"
440 PRINT "@.@.@.@@@.@@.@@@@@.@@@.@.@.@.@.@"
450 PRINT "@.@.@....................@.@.@.@"
460 PRINT "@.@.@.@@@.@.@.@.@@@@@@@.@@...@.@"
470 PRINT "@...@.@...@..............@@@.@.@"
480 PRINT "@.@@@.@.@@@@@@@@@@@@.@@@.@@@.@.@"
```

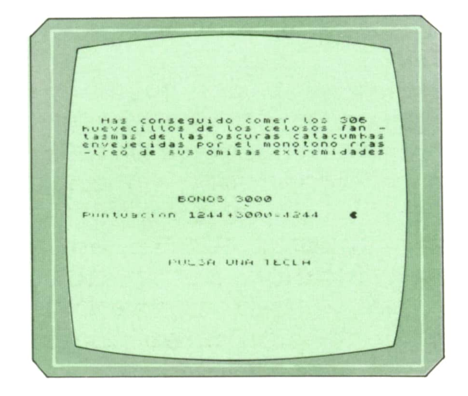
```
 490 PRINT "@.............................@"
 500 PRINT "@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@"
 510 PRINT AT 21,15;"PUNTOS= ";pu
 520 LET x=16
 530 LET y=16
 540 LET e$=""
 550 LET w$=""
 560 LET q$=""
 570 LET x1=10
 580 LET y1=1
 590 LET x2=15
 600 LET y2=4
 610 LET x3=11
 620 LET y3=18
 630 LET x4=1
 640 LET y4=18
 650 PRINT AT y,x;CHR$ 144
 660 POKE 23658,8
 670 LET k$=INKEY$
 680 IF k$="Q" THEN  IF SCREEN$ (y,x-1)<>"@" THEN  LET x=x-1: PRINT AT y,x+1;" "
: LET a$=SCREEN$ (y,x): PRINT AT y,x;CHR$ 145: IF a$="." THEN  LET pu=pu+10
 690 IF k$="W" THEN  IF SCREEN$ (y,x+1)<>"@" THEN  LET x=x+1: PRINT AT y,x-1;" "
: LET a$=SCREEN$ (y,x): PRINT AT y,x;CHR$ 144: IF a$="." THEN  LET pu=pu+10
 700 IF k$="O" THEN  IF SCREEN$ (y-1,x)<>"@" THEN  LET y=y-1: PRINT AT y+1,x;" "
: LET a$=SCREEN$ (y,x): PRINT AT y,x;CHR$ 146: IF a$="." THEN  LET pu=pu+10
 710 IF k$="K" THEN  IF SCREEN$ (y+1,x)<>"@" THEN  LET y=y+1: PRINT AT y-1,x;" "
: LET a$=SCREEN$ (y,x): PRINT AT y,x;CHR$ 147: IF a$="." THEN  LET pu=pu+10
 720 IF pu=3060 THEN  GO TO 1170
 730 GO SUB 820
 740 GO SUB 870
 750 GO SUB 950
 760 IF ATTR (y,x)<>7 THEN  GO SUB 1030
 770 PRINT AT 21,0;v$;AT 21,23;pu
 780 GO TO 660
 785 REM
 790 REM ***********************
 800 REM ***** MARCIANO-1 ******
 810 REM ***********************
 815 REM
 820 LET i=SGN (x-x1)
 830 LET a=SGN (y-y1)
 840 IF SCREEN$ (y1,x1+i)<>"@" THEN  LET x1=x1+i: PRINT AT y1,x1-i;q$: LET q$=SC
REEN$ (y1,x1): PRINT AT y1,x1; INK 1;CHR$ 148
 850 IF SCREEN$ (y1+a,x1)<>"@" THEN  LET y1=y1+a: PRINT AT y1-a,x1;q$: LET q$=SC
REEN$ (y1,x1): PRINT AT y1,x1; INK 1;CHR$ 148
 860 RETURN
 865 REM
 870 REM ***********************
 880 REM ***** MARCIANO-2 ******
 890 REM ***********************
 895 REM
 900 LET i=SGN (x-x2)
 910 LET a=SGN (y-y2)
 920 IF SCREEN$ (y2,x2+i)<>"@" THEN  LET x2=x2+i: PRINT AT y2,x2-i;w$: LET w$=SC
REEN$ (y2,x2): PRINT AT y2,x2; INK 2;CHR$ 148
 930 IF SCREEN$ (y2+a,x2)<>"@" THEN  LET y2=y2+a: PRINT AT y2-a,x2;w$: LET w$=SC
REEN$ (y2,x2): PRINT AT y2,x2; INK 2;CHR$ 148
 940 RETURN
 945 REM
 950 REM ***********************
 960 REM ***** MARCIANO-3 ******
 970 REM ***********************
 975 REM
 980 LET i=SGN (x-x3)
 990 LET a=SGN (y-y3)
1000 IF SCREEN$ (y3,x3+i)<>"@" THEN  LET x3=x3+i: PRINT AT y3,x3-i;e$: LET e$=SC
REEN$ (y3,x3): PRINT AT y3,x3; INK 4;CHR$ 148
1010 IF SCREEN$ (y3+a,x3)<>"@" THEN  LET y3=y3+a: PRINT AT y3-a,x3;e$: LET e$=SC
```

```
REEN$ (y3,x3): PRINT AT y3,x3; INK 4;CHR$ 148
1020 RETURN
1025 REM
1030 REM ************************
1040 REM ******* MUERTO ********
1050 REM ************************
1055 REM
1060 PRINT AT 10,6; FLASH 1;" * * M U E R T O * * "
1065 REM
1070 FOR f=30 TO 20 STEP -5
1080    BEEP .5,F
1090 NEXT f
1100 LET vi=vi-1
1110 IF vi<0 THEN  GO TO 1750
1120 LET v$=v$( TO vi)
1130 PRINT AT 21,0; FLASH 1;"         PULSA UNA TECLA
1140 IF INKEY$="" THEN  GO TO 1140
1150 CLS
1160 GO TO 260
1165 REM
1170 REM ***********************
1180 REM *** FIN DE PANTALLA ***
1190 REM ***********************
1195 REM
1200 CLS
1210 LET b=31
1220 LET a$=CHR$ 16+CHR$ 2+CHR$ 148+" "+CHR$ 16+CHR$ 4+CHR$ 148+" "+CHR$ 16+CHR$
 3+CHR$ 148+" "
1230 FOR f=26 TO 3 STEP -1
1240    PRINT  INK 1;AT 10,f;a$
1250    IF f<18 THEN  PRINT AT 10,b;CHR$ 145+"  ": LET b=b-2
1260    IF b<7 THEN  LET a$=a$( TO LEN a$-4)
1270    FOR x=1 TO 5: NEXT x
1280 NEXT f
1290 FOR f=3 TO 27
1300    PRINT AT 10,f; INK 6;" "+CHR$ 144
1310    BEEP .005,30
1320 NEXT f
1330 FOR f=10 TO 15
1340    PRINT AT f,28; INK 6;CHR$ 147;AT f-1,28;" "
1350    BEEP .005,50
1360 NEXT f
1370 LET a=144
1380 FOR f=1 TO 21
1390    PRINT AT 15,28; INK 6;CHR$ a
1400    LET a=a+1
1410    IF a=148 THEN  LET a=144
1420    FOR x=1 TO 4
1430    NEXT x
1440    BEEP .05,-50+INT (RND*100)
1450 NEXT f
1460 PRINT AT 3,0;"  Has conseguido comer los 306  huevecillos de los celosos fa
n -tasmas de las oscuras catacumbasenvejecidas por el monotono rras-treo de sus
omisas extremidades"
1470 PRINT AT 13,10; FLASH 1;"BONOS ";vi*1000
1480 PRINT AT 15,0;"Puntuacion ";pu;"+";vi*1000;"=";pu+vi*1000
1490 PRINT AT 21,9; FLASH 1;"PULSA UNA TECLA"
1500 IF INKEY$="" THEN  GO TO 1500
1510 FOR f=15 TO 0 STEP -1
1520    PRINT AT f,28; INK 6;CHR$ 146;AT f+1,28;" "
1530 NEXT f
1540 FOR f=28 TO 0 STEP -1
1550    PRINT AT 0,f; INK 6;CHR$ 145+" "
1560 NEXT f
1570 PRINT AT 0,0;" ";AT 1,0; INK 6;CHR$ 144
1580 IF INKEY$="" THEN  GO TO 1580
1590 LET a=7: LET b=3
1600 PRINT AT 1,0;" "
```

```
1610 FOR w=1 TO 2
1620    FOR c=b TO a
1630      FOR f=0 TO 30
1640        PRINT AT c,f; INK 6;" "+CHR$ 144
1650      NEXT f
1660      PRINT AT c,31;" "
1670    NEXT c
1680    LET a=21
1690    LET b=13
1700 NEXT w
1710 CLS
1720 LET p=p+pu
1730 LET pu=0
1740 GO TO 260
1745 REM
1750 REM ***********************
1760 REM ** FINAL DE PARTIDA ***
1770 REM ***********************
1775 REM
1780 CLS
1790 PRINT AT 10,2;"Tu puntuacion ha sido= ";p
1800 PRINT '"  Lastima,parecias bueno."
1810 PRINT "  No te preocupes,cuenta con mi compasion."
1820 PRINT #0; FLASH 1;"   QUIERES JUGAR OTRA (S/N) ?   "
1830 LET k$=INKEY$
1840 IF k$="S" OR k$="s" THEN  GO TO 200
1850 IF k$="N" OR k$="n" THEN  GO TO 2030
1860 GO TO 1830
1865 REM
1870 REM ***********************
1880 REM ****** GRAFICOS *******
1890 REM ***********************
1895 REM
1900 RESTORE 1980
1910 FOR f=97 TO 101
1920    FOR c=0 TO 7
1930      READ a
1940      POKE USR (CHR$ f)+c,a
1950    NEXT c
1960 NEXT f
1970 RETURN
1980 DATA 0,28,62,124,120,124,62,28
1990 DATA 0,56,124,62,30,62,124,56
2000 DATA 0,34,119,127,127,62,28,0
2010 DATA 0,56,124,254,254,238,68,0
2020 DATA 56,124,214,254,238,254,170,170
```

Cada vez que termines una pantalla después de comerte los 306 puntos que la componen, según el tiempo que hayas tardado en hacerlo, el ordenador te regalará más o menos puntos.

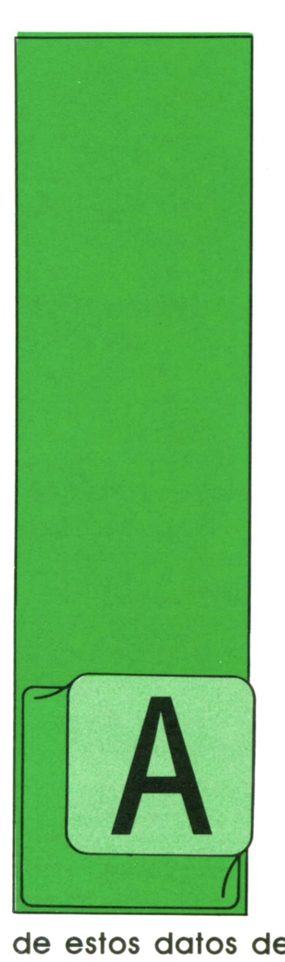




El programa nos da bonos cada vez que completamos una pantalla.

TECNICAS DE ANALISIS

DOCUMENTOS DE SALIDA: PREIMPRESOS



SI como la toma de datos es básica para que las informaciones que procesa el ordenador sean válidas igual que los resultados obtenidos, la presentación final de estos datos de salida es muy importante, para que sean de verdad útiles y se obtenga toda la eficacia que se espera del proceso informático realizado.

Los documentos de salida de una aplicación informática son como la «tarjeta de visita» o «cara externa» del sistema y, por tanto, han de cuidarse también en su aspecto general.

Es decir, básicamente dos son los aspectos a considerar en el diseño de un documento de salida: el aspecto general (formato, presentación, tamaño, distribución de las informaciones, etc.) y contenido (seleccionado, presentado de un modo claro, ordenado, etc.).

Respecto del primero de estos aspectos, hay que tener en cuenta varias características:

a) **Preimpresión del papel.** Se llama preimpreso a un documento en que están impresas, antes de su paso por la impresora del ordenador, un conjunto de informaciones: cabeceras, rótulos fijos, líneas de recuadros, dibujos, etc.

La presentación de un documento preimpreso es más agradable que si toda la información se escribe en la impresora del ordenador (1).

El diseño e impresión de un documento preimpreso, sin embargo, suelen ser caros, por lo que normalmente no está justificada en un entorno de informática personal (ni, en ocasiones, a nivel profesional; aunque en este caso intervienen otros factores de imagen, prestigio, etc.).

Además, la preparación de la impresión es más complicada. En efecto, a poco contenido que tenga el documento preimpreso, hay que ajustar cuidadosamente el papel para que cada dato se escriba en su lugar correspondiente: los programas o rutirnas que han de imprimir los resultados suelen tener previsto al comienzo un bucle que escribe una hoja simulada completa (con datos falsos) y pregunta si el impreso está ya ajustado (en altura y a derecha e izquierda); mientras se conteste que no, se sigue repitiendo el bucle hasta que el papel queda perfectamente ajustado. Este proceso es

---

(1) Nos estamos refiriendo a las impresoras convencionales (de matriz de puntos, de líneas, de chorro de tinta, etc.); un caso aparte lo constituyen las "impresoras láser" en las que se suele imprimir no sólo los textos y números variables sino las informaciones permanentes y hasta dibujos, firmas, etc., sin pérdida de velocidad. Sin embargo, este tipo de impresoras es aún caro comparativamente con el resto de las impresoras de ordenador, aunque hoy en día hay máquinas láser pequeñas para su conexión a ordenador de tipo profesional (PC y compatibles) a un precio módico (relativamente). En cualquier caso, una impresora láser sólo se justifica si se han de imprimir un número considerable de documentos y/o éstos se pretende que tengan gran calidad.

muy útil sobre todo si el preimpreso es en papel continuo, pues una vez ajustado queda ya fijo durante toda la impresión a realizar (hasta que se cambia de modelo de papel): si el papel es de hojas sueltas, se suele poner en él una marca para ajustarlo en la impresora (con la escala numerada que las impresoras suelen tener) al ir introduciendo cada hoja.

Otra dificultad que existe cuando se prepara un documento en papel preimpreso es que no se puedan introducir modificaciones: en efecto, aparece un conflicto entre el interés de que cada hoja preimpresa resulte barata (haciendo una gran tirada; a «consumir» o utilizar, por tanto, en un período largo de tiempo) y la flexibilidad necesaria para poder introducir cambios (sin tener que desperdiciar una cantidad grande de hojas preimpresas que ya se tengan preparadas). Una pequeña modificación o la corrección de un error que puede suponer un problema menor a nivel de programa, es una dificultad grande (costosa, sobre todo) cuando se tiene preimpresa una cantidad grande de papel. De hecho ésta es una de las mayores dificultades del uso de preimpresos en los centros de proceso de datos, sobre todo en las aplicaciones de nueva implantación (durante la fase de control y ajuste de los procesos).

Por otro lado, hay que tener en cuenta que hay muchos tipos diferentes de preimpresos; en hoja suelta o en papel continuo, en papel con copia (papel de calco o «autocopiativo») o sólo original (escribiendo, incluso en ocasiones, varias veces cada página para tener diversos originales, en vez de tener original y copia) o tipos especiales de preimpresos: sobre completo de contenido fijo en el que se escriben, solamente, los datos del destinatario; sobre con una «banda de ocultación» (zona donde están preimpresos de un modo anárquico cantidad de caracteres y números que se sobreponen para que no se lea en el sobre lo escrito por el ordenador, aunque sí se lee en la copia autocopiativa que se encuentra en su interior), etc.

Sin embargo, a pesar de todas las dificultades expuestas, los preimpresos son ampliamente utilizados. Aparte el hecho de que proporcionan un aspecto más «profesional» al impreso resultante (razón básica para su uso en la mayoría de los casos), proporcionan algunas ventajas adicionales:

a) el coste puede ser menor si se hacen grandes tiradas: en efecto, la escritura de los textos fijos de un impreso suele costar de 2 a 4 veces menos en un preimpreso que si se hace con la impresora del ordenador (dependiendo de la cantidad de texto a escribir, de su distribución en la página, etc., y del tipo de equipo informático —ordenador e impresora— que se esté utilizando);

b) la «riqueza» gráfica es mayor, pues excepto que se utilicen impresoras gráficas de coste más alto y velocidad más baja, con un ordenador sólo se pueden representar los caracteres alfabéticos, los números y unos cuantos símbolos especiales, junto con rayas y puntos. Por el contrario, en el preimpreso se puede imprimir cualquier gráfico, símbolo, anagrama, etc. (e incluso imprimir a varios colores);

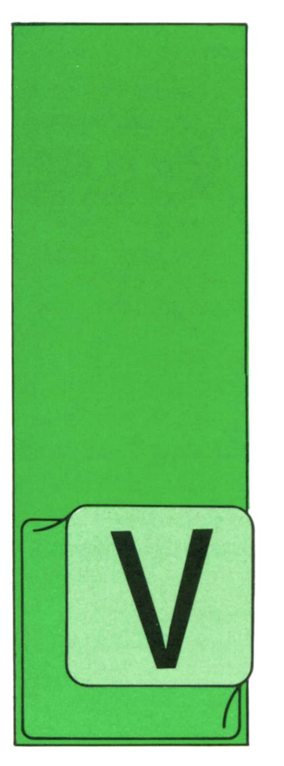
c) ahorro de tiempo de máquina. En ocasiones el gasto de tiempo de ordenador es una dificultad no sólo por el coste (ya comentado), sino porque se ocupa el equipo en una tarea evitable;

d) ahorro de espacio. La variedad tipográfica es mayor en una imprenta que en la impresora del ordenador: por ello, los comentarios, explicaciones, instrucciones de uso, claves, etc., se pueden imprimir en letra menor que la de la impresora del ordenador. Por otro lado, las líneas de separación o los recuadros de enmarcar textos se puede prever que vayan entre dos líneas o dos caracteres de los que va a imprimir el ordenador;

e) claridad en la distribución de los datos, no sólo porque se pueden introducir más tipos de líneas, cuadros, etc., sino porque se pueden incluir sombreados, rayados, etc., que si se realizan con el ordenador ensombrecen, en cantidad de ocasiones, más que aclaran los datos a observar.

En función de todos los elementos presentados es importante, especialmente desde el punto de vista económico, la decisión de hacer o no un preimpreso y, caso que se decida imprimirlo, aprovechar todas sus posibilidades para mejorar la claridad y cantidad de datos a presentar así como la facilidad de explotación.

TECNICAS DE PROGRAMACION





Instrucciones condicionales

VEAMOS ahora cómo se escriben las instrucciones condicionales en el lenguaje PASCAL. Al igual que en BASIC, se utilizan las palabras reservadas inglesas IF, THEN, ELSE, para construirlas. Por tanto, la forma general de la instrucción condicional será:

```
IF condición
   THEN acción-1
   ELSE acción-2;
```

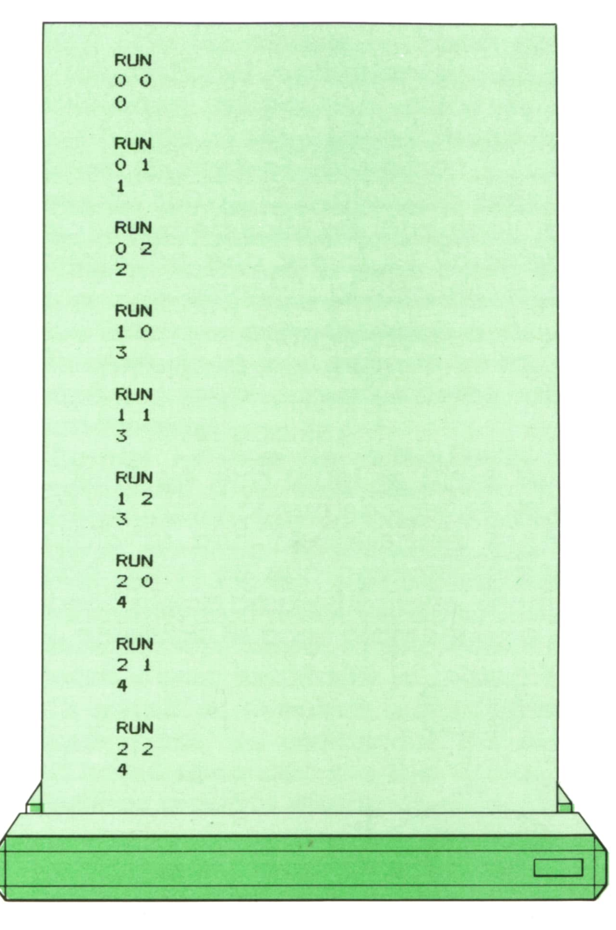
donde acción-1 y acción-2 pueden ser instrucciones de cualquier tipo: bloques secuenciales, instrucciones condicionales, bucles, asignaciones de valor, etc.

La única diferencia con relación al BASIC es el punto y coma que hay que colocar en PASCAL al final de todas las instrucciones.

Veamos cómo se escribe en PASCAL el ejemplo BASIC que vimos en el capítulo anterior:

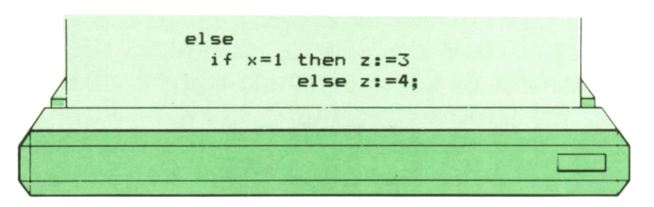
```
program EJEMPLO;
var
  x, y, z: integer;
begin
  readln(x,y);
  if x=0 then
            if y=0 then z:=0
                     else
                       if y=1 then z:=1
                                else z:=2
         else
            if x=1 then z:=3
                     else z:=4;
  writeln (z);
end.
```

cuyo organigrama y funcionamiento es prácticamente idéntico:



Observando los puntos y comas, que señalan el final de cada instrucción, puede verse que, al igual que el programa BASIC, el programa PASCAL tiene tres instrucciones ejecutables, una de las cuales es muy larga. Veámosla en forma aislada:

```
if x=0 then
      if y=0 then z:=0
               else
                 if y=1 then z:=1
                          else z:=2
```



A esta instrucción le corresponde exactamente el mismo organigrama que a la instrucción BASIC correspondiente.

También en PASCAL puede ocurrir que la instrucción condicional sea incompleta, careciendo de cláusula ELSE:

```
IF condición
  THEN acción-1;
```

donde acción-1 puede ser una instrucción de cualquier tipo. En particular, podría ser otra instrucción condicional. ¿Qué ocurrirá si esta vez se trata de un IF-THEN-ELSE completo? Es decir, si tenemos una sola cláusula ELSE y dos cláusulas THEN, ¿con cuál de ellas debe asociarse la primera? La forma que adoptaría la instrucción sería la siguiente:

```
IF condición-1
  THEN IF condición-2
      THEN acción-1
      ELSE acción-2;
```

Pues bien: al igual que en BASIC, en PASCAL se supone que la cláusula ELSE va siempre emparejada con la cláusula THEN más próxima que no tenga pareja. Por tanto, el organigrama correspondiente a este ejemplo sería el siguiente:

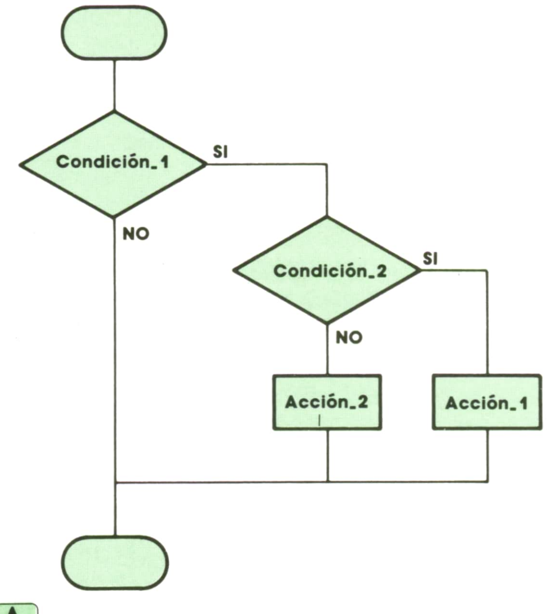




Fig. 1.

y es el primer THEN de la instrucción el que se queda sin pareja.

Si quisiéramos forzar al compilador a adoptar la opción contraria, emparejando la cláusula ELSE con la primera cláusula THEN, en lugar de la segunda, podríamos conseguirlo así:

```
IF condición-1
  THEN BEGIN
      IF condición-2
        THEN acción-1
      END
  ELSE acción-2;
```

cuyo organigrama es:

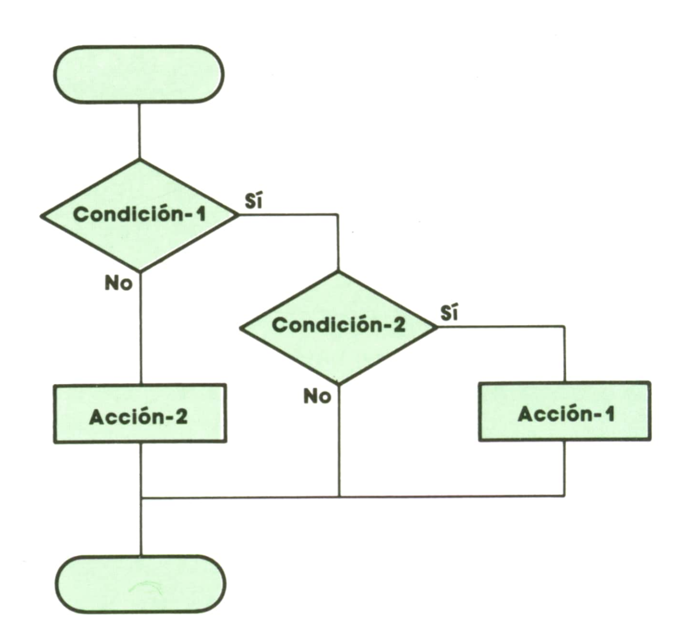
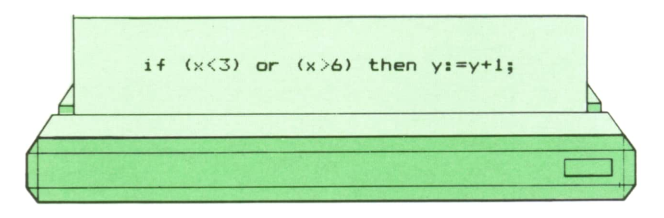




Fig. 2.

Veamos ahora un ejemplo de una instrucción condicional PASCAL cuya condición sea algo más compleja que las anteriores:



Esta línea suma 1 al valor de la variable y si el valor de la variable x es menor que 3 o mayor que 6.

Por último, veamos cómo se puede introducir un bloque secuencial en una instrucción condicional PASCAL:

```
if i=68 then
     begin
       x:=x+1;
       y:=y+1;
       writeln(x,' ',y)
     end
   else
     begin
       x:=x-1;
       y:=y-1;
       writeln(y,' ',x)
     end;
```

donde la misma instrucción IF-THEN-ELSE contiene dos bloques, uno en la parte THEN y otro en la parte ELSE. Veamos el organigrama correspondiente:

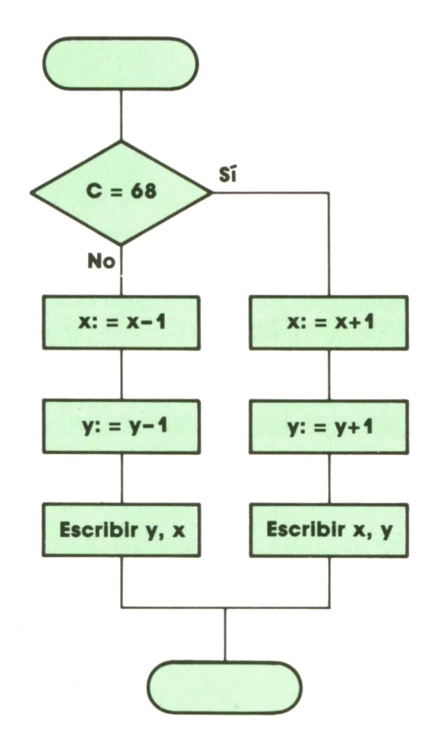
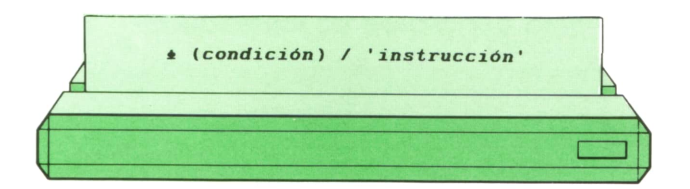



Fig. 3.

En el lenguaje APL no existe la instrucción IF-THEN-ELSE como tal. En primer lugar, no hay palabras reservadas en inglés (ni en ninguna otra lengua), sino sólo símbolos. En segundo lugar, el control de la marcha de los programas no se especifica mediante las instrucciones típicas de la programación estructurada, sino mediante la transferencia generalizada, que estudiaremos más adelante.

Sin embargo, no es difícil simular la estructura IF-THEN en APL, mediante la siguiente construcción:

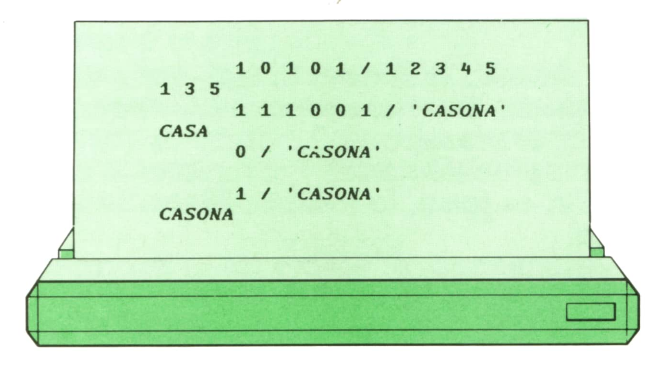


que podemos leer: «Si condición, entonces instrucción». Obsérvese, sin embargo, que la instrucción a realizar debe especificarse entre comillas. La razón de esto es la siguiente: en APL, una condición es una expresión lógica, cuyo valor será cero (si la expresión es verdadera) o uno (si la expresión es falsa).

La operación «selección», representada por una línea inclinada (/) realiza la siguiente acción: a su izquierda debe haber una serie de ceros y unos, y a su derecha, una serie cualquiera de datos numéricos o alfabéticos. Las dos series deben tener la misma longitud. El resultado de esta operación es una nueva serie, del mismo tipo que la serie de la derecha, donde sólo aparecerán los elementos de ésta que correspondan a los unos de la serie de la izquierda, mientras que los que correspondan a los ceros habrán desaparecido.

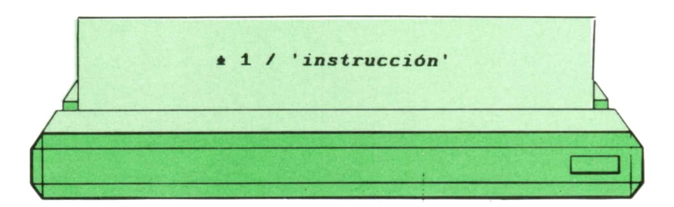
Además, la serie de la izquierda puede reducirse a un solo elemento, independientemente de los que tenga la serie de la derecha. Se entiende que este elemento (un 1 o un 0) se extenderá automáticamente a todos los de la serie de la derecha. Por tanto, la expresión 1/serie dará como resultado la misma serie de la derecha, mientras que 0/serie eliminará todos los elementos de ésta y dará como resultado la serie vacía.

Veamos algunos ejemplos:

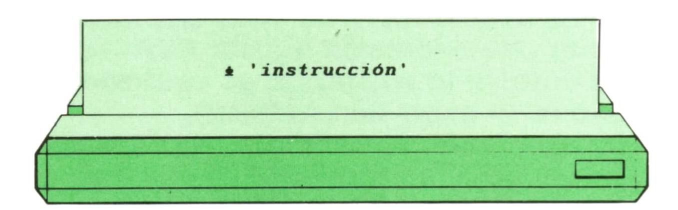


Por tanto, la instrucción que simula la construcción IF-THEN se reduce en la práctica a uno de los dos casos siguientes:

1. Si la condición es verdadera:

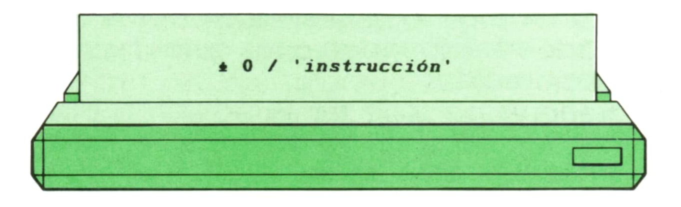


que se reduce a:

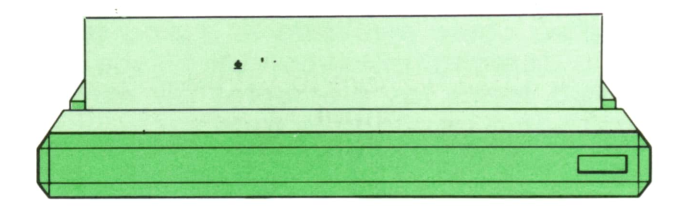


El símbolo situado a la izquierda de la expresión anterior ejecuta la instrucción expresada entre comillas. Por tanto, si la condición es verdadera, la instrucción se ejecuta.

2. Si la condición es falsa:



que se reduce a:

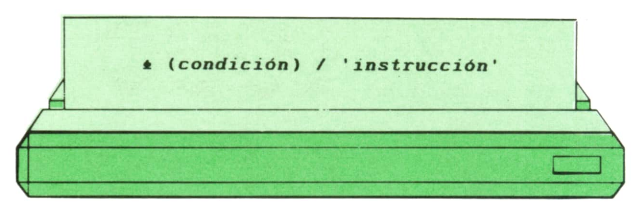


El símbolo situado a la izquierda de la expresión anterior ejecuta la instrucción vacía expresada entre comillas (es decir, no ejecuta nada). Por tanto, si la condición es falsa, la instrucción no se ejecuta.

Resumiendo, el efecto de la expresión anterior es el siguiente:

1. Si la condición es verdadera, la instrucción dada entre comillas se ejecuta.
2. Si la condición es falsa, la instrucción dada entre comillas no se ejecuta.

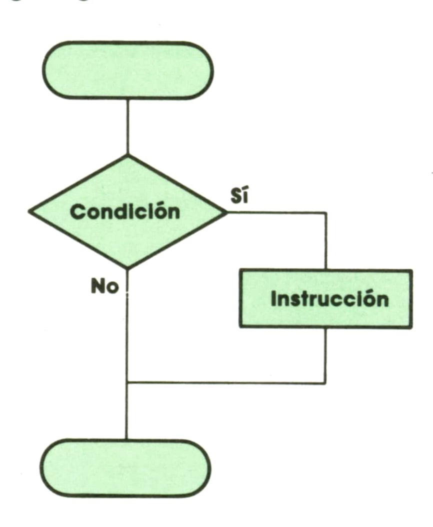
Por tanto, la expresión:



equivale en APL a

IF condición THEN instrucción

cuyo organigrama es:



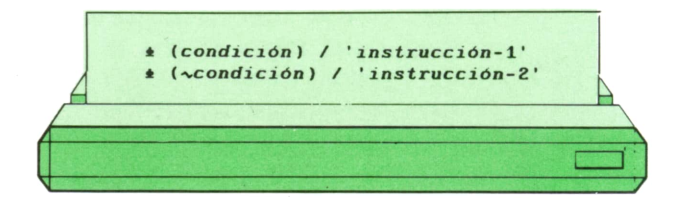
La construcción IF-THEN-ELSE no es tan fácil de simular en APL. Puede conseguirse, sin embargo, mediante la construcción equivalente siguiente:

IF condición THEN instrucción-1
IF NOT condición THEN instrucción-2

donde la condición de la segunda instrucción es la negación de la condición de la primera. Estas dos instrucciones, en conjunto, actúan exactamente igual que la siguiente:

IF condición THEN instrucción-1 ELSE instrucción-2

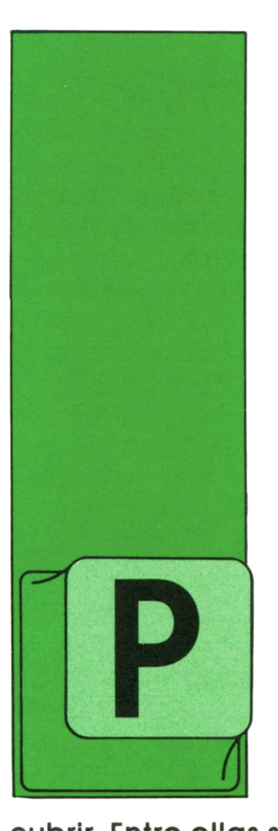
En APL, esta construcción se representaría así:



donde la condición es idéntica en ambas instrucciones, excepto que en la segunda está precedida por un símbolo (la tilde) que en APL representa la negación lógica.

LOGO

QUÉ ES UNA VARIABLE



POCO a poco vamos conociendo todas las cosas que tiene o que puede hacer la tortuga y para qué nos pueden servir. Pero todavía nos quedan algunas por descubrir. Entre ellas se encuentran las variables.

Nuestra amiga dispone de una serie de cajones (como los de un armario) dentro de los cuales puede guardar valores, lo mismo que dentro de una hucha guardamos monedas o en una estantería ponemos libros, cuadernos, etc.

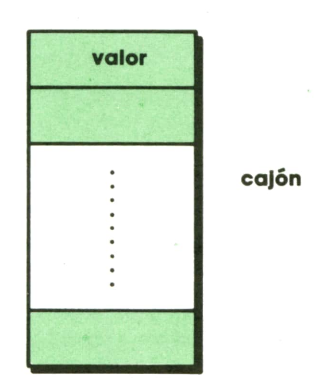



Fig. 1.

Estos valores pueden ser números, palabras, e incluso frases,




Fig. 2.

y los podemos ir cambiando según nos vaya interesando. Por este motivo, cada uno de estos cajones se dice que es una VARIABLE, ya que su contenido no es un valor fijo que siempre sea igual, sino que puede ir variando.

Como es lógico, para poder meter un valor en un determinado cajón necesitamos distinguirlo de los demás. La mejor forma de lograr esto es dando un nombre a cada uno de los cajones que vayamos utilizando. Por ejemplo, los tres cajones anteriores podrían llamarse:




Fig. 3.

Como este nombre lo escogemos nosotros, lo más normal es usar nombres que signifiquen algo para nosotros, es decir, que nos indiquen qué es lo que va a ir conteniendo ese cajón o para qué lo vamos a utilizar.

Una vez que tenemos un cajón que contiene algo hemos de diferenciar su nombre y su valor. Para ello, siempre que queramos hacer referencia al nombre de un cajón usaremos,

«nombre

mientras que si lo que nos interesa es el valor (el contenido) de ese cajón pondremos

:nombre

En el caso de los tres cajones anteriores, tendríamos que:

Nombre	Contenido
“EDAD “AMIGO “SALUDO	:EDAD = 5 :AMIGO = JULIAN :SALUDO = BUENOS DIAS

Supongamos que queremos tener un cajón en el que vamos a ir guardando los diferentes días de la semana. Tendremos que:

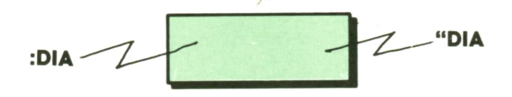



Fig. 4.

“DIA = Nombre del cajón.
:DIA = Contenido del cajón.

— LUNES
— MARTES
— MIERCOLES
— JUEVES
— VIERNES
— SABADO
— DOMIGO



Os proponemos

1. Piensa el nombre, el contenido o los valores posibles de las siguientes variables:

Nombre	Contenido	Valores posibles
“PUNTOCARDINAL	:PUNTOCARDINAL	NORTE, SUR, ESTE, OESTE
“AMIGO	:ARCOIRIS	0, 1, 2, 3, 4, 5, 6, 7, 8, 9
“MES		NIEVE, LLUVIA, VIENTO

2. Intenta descubrir el valor que se almacenaría en el cajón 3 después de hacer las siguientes operaciones:

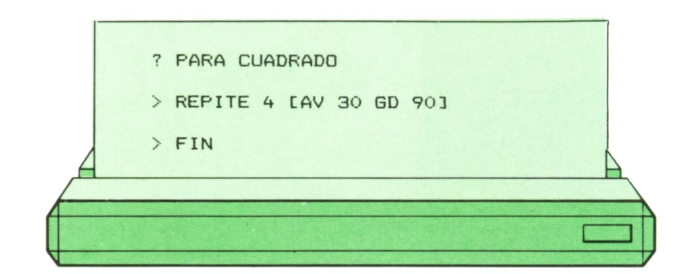
a. En “cajón1 se guarda 7.
b. En “cajón2 se guarda :cajón1 + 3.
c. En “cajón3 se guarda :cajón2 + :cajón1.



Utilidad de las variables

Ya sabemos que nuestra amiga la tortuga tiene una memoria que le permite recordar todas aquellas cosas nuevas que le vayamos enseñando mediante la definición de procedimientos.

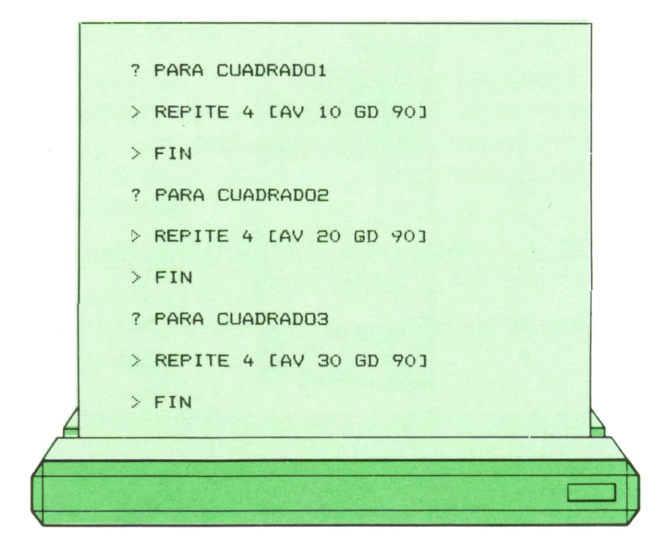
Supongamos que le hemos enseñado a hacer un cuadrado:



Siempre que le mandemos a la tortuga que ejecute este procedimiento, nos dibujará un cuadrado del mismo tamaño (30).

En caso de que lo queramos hacer más grande o más pequeño, tendremos que utilizar e; editor para borrar el 30 y poner el valor que nos vaya interesando en cada momento.

Otra solución es tener definidos varios procedimientos que dibujen cuadrados de distintos tamaños:



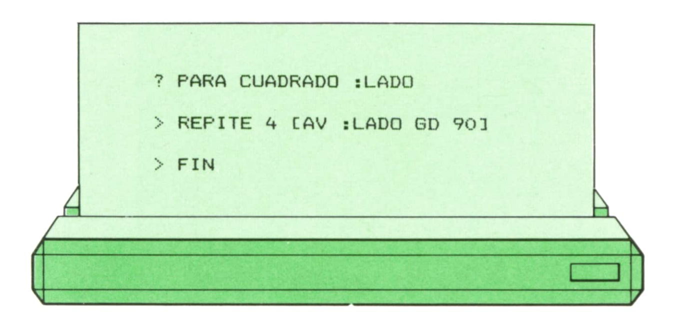
Como vemos, estos dos métodos no son muy buenos, ya que con el primero tenemos que estar utilizando continuamente el editor, y con el segundo tenemos ocupada la memoria de la tortuga con un montón de procedimientos que

hacen prácticamente lo mismo. En este caso, lo que más nos interesa es utilizar una variable.

Al igual que los comandos AV o GD siempre llevan una entrada, es decir, van acompañados de un número que indica el número de pasos, o de grados que ha de dar la tortuga, los procedimientos también pueden llevar entradas, cuyo valor se lo daremos cuando la tortuga vaya a ejecutarlo.

En el caso del cuadrado, nos interesa tener una variable en la que vayamos guardando el tamaño del cuadrado a dibujar.

La forma de definir un procedimiento que dibuje cuadrados de tamaño variable es:



```
? PARA CUADRADO :LADO
> REPITE 4 [AV :LADO GD 90]
> FIN
```

Ahora nos queda decirle a la tortuga el valor que queremos almacenar en la variable. Así, si vamos a dibujar un cuadrado de tamaño diez diremos

?CUADRADO 10

y si después lo queremos de tamaño 25 pondremos

?CUADRADO 25

De esta manera no tenemos que entrar en el editor cada vez que queramos cambiar el tamaño del cuadrado, sino que basta con dar un valor a la variable cuando vayamos a ejecutar el procedimiento. Y, por otro lado, la tortuga no tiene que memorizar varios procedimientos que hagan cuadrados, sino que sólo tiene que recordar uno.

Algunos dibujos con cuadrados de tamaño variable

Una vez que sabemos dibujar cuadrados de lado variable, resulta muy sencillo realizar figuras en las que intervengan éstos con diferentes tamaños.

Como es lógico, lo primero que tendremos que hacer será enseñar a la tortuga el procedimiento que dibuje cuadrados de varios tamaños.

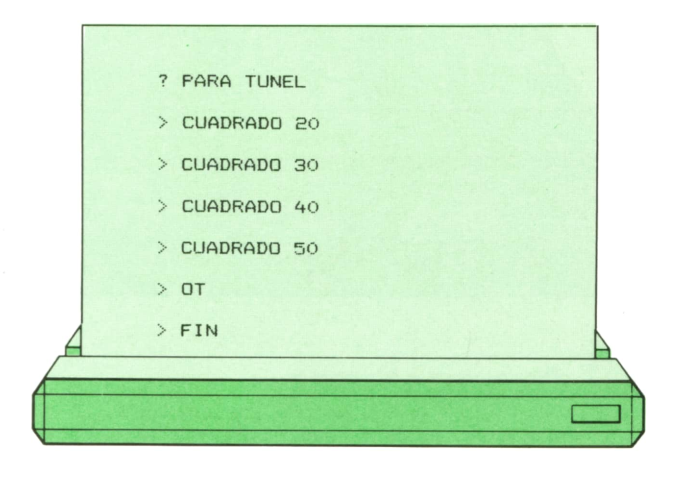
Supongamos que ahora queremos pintar la siguiente figura:





Fig. 5.

El procedimiento correspondiente sería:



```
? PARA TUNEL
> CUADRADO 20
> CUADRADO 30
> CUADRADO 40
> CUADRADO 50
> OT
> FIN
```

También podemos dibujar la siguiente columna:

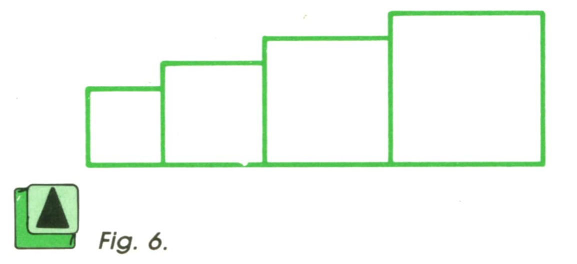
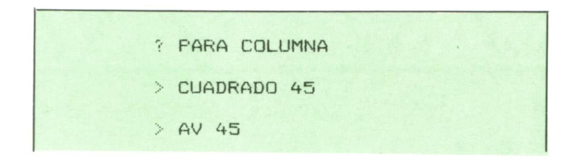
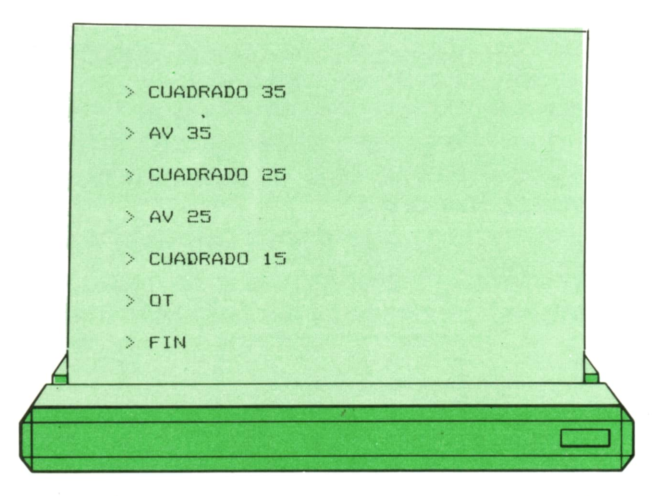


Fig. 6.

con este procedimiento:



```
? PARA COLUMNA
> CUADRADO 45
> AV 45
```



Por último, podemos obtener esta figura

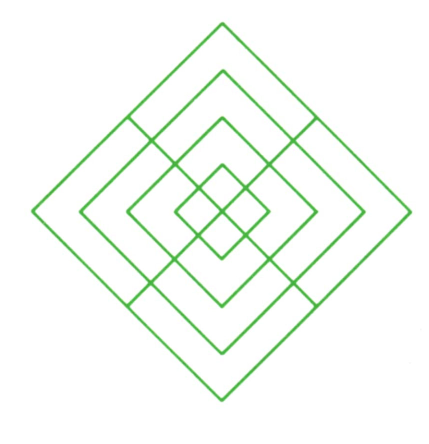
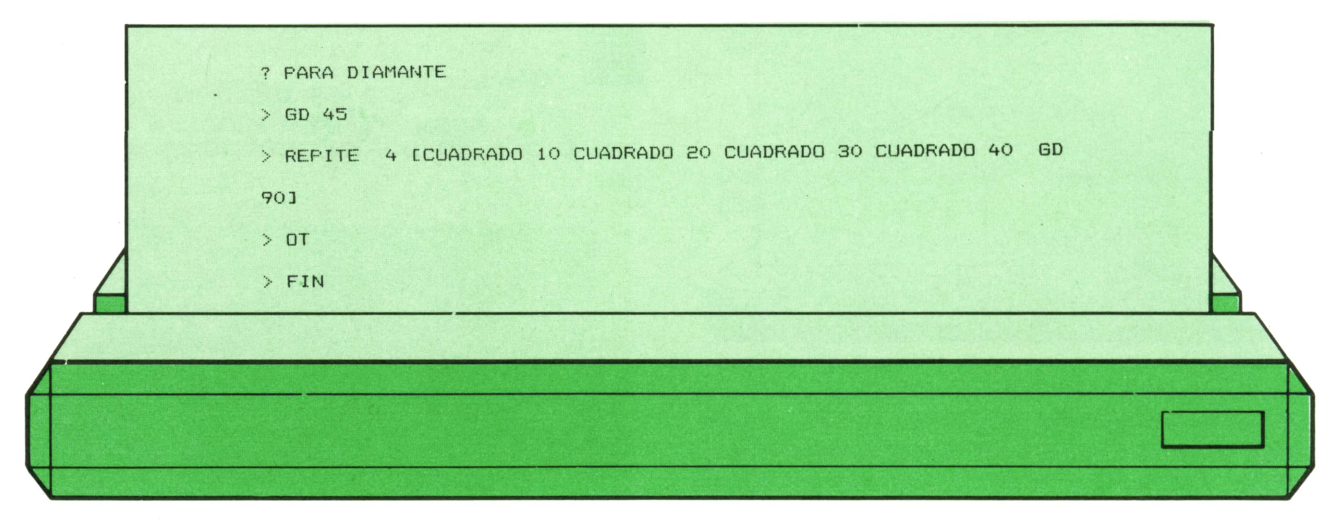




Fig. 7.

ejecutando el procedimiento DIAMANTE



y luego esta cadena





Fig. 9.



Os proponemos

1. Intenta definir los procedimientos que dibujan las siguientes figuras geométricas con tamaño variable:

- Triángulo
- Pentágono
- Hexágono
- Octógono
- Círculo

2. Usando el procedimiento que dibuja triángulos de distinto tamaño, pinta estas montañas:

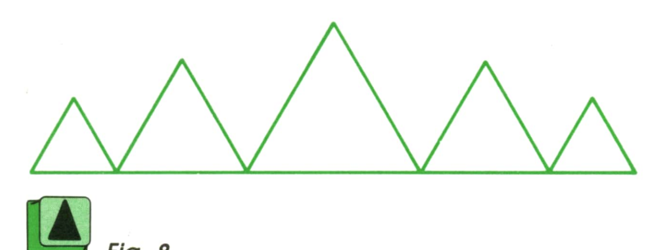


Fig. 8.

3. Ahora puedes pintar este túnel:





Fig. 10.

4. Define los procedimientos para obtener estas figuras:

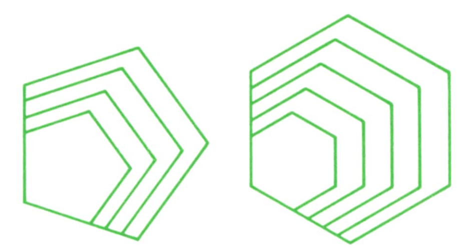




Fig. 11.

5. Pinta esta serie de circunferencias:

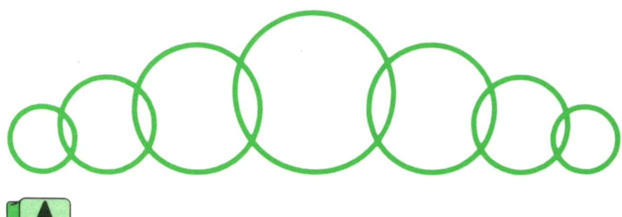
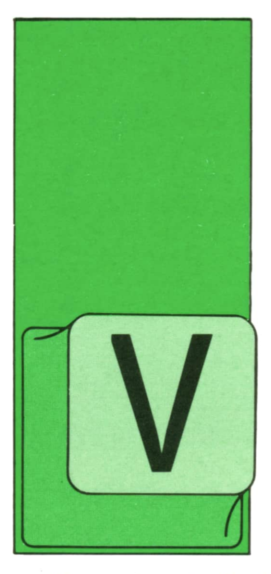




Fig. 12.

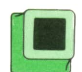
PASCAL



AMOS a hacer nuestra primera incursión en el terreno de los programas que hacen gráficos. Desafortunadamente, como las posibilidades gráficas varían con cada ordenador (e incluso muchos de ellos no son capaces de manejar gráficos), en el Pascal estándar no hay nada definido al respecto y, por tanto, la forma de trabajar varía de unos compiladores a otros.

Ante esta situación, no quedan más que dos alternativas, o bien nos abstenemos de hacer gráficos, o bien optamos por un compilador concreto, y es esta última la que vamos a escoger, optando por el Turbo-Pascal de la casa Borland, que funciona en los ordenadores personales IBM y compatibles y que es uno de los más baratos del mercado.

No obstante, veremos que, gracias a los procedimientos y funciones, es posible estructurar los programas de manera que las zonas susceptibles de cambio queden claramente delimitadas.



Las figuras de Hilbert

Observemos estas figuras:

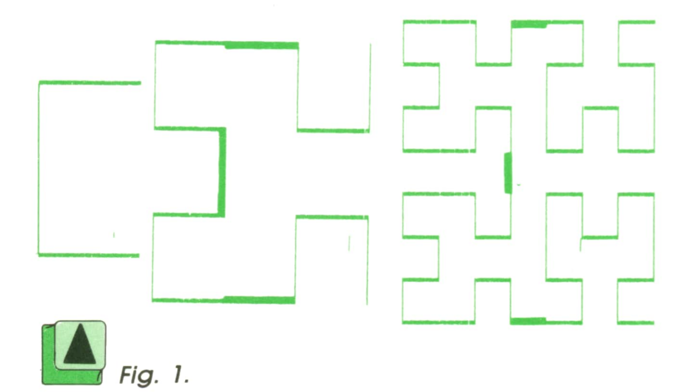


Fig. 1.

Salta a la vista que la de la derecha está formada por cuatro figuras como la del centro giradas adecuadamente y unidas por tres rectas. Análogamente, la del medio está formada por cuatro como la de la izquierda unidas de idéntica manera entre sí.

Si llamamos H_1 a la más simple, H_2 a la siguiente y H_3 a la de la derecha, podríamos generalizar la descripción de las figuras diciendo que la curva H_{i+1} se forma a partir de cuatro del tipo H_i unidas entre sí; incluso, podríamos decir que H_1, la de la izquierda, está formada por cuatro figuras del tipo H_0, o «figura inexistente» unidas entre sí por los tres tramos correspondientes.

La figura H_N se denomina «figura de Hilbert de nivel N» en honor a su inventor D. Hilbert.

Supongamos que las figuras las pintamos con un lápiz; el procedimiento para trazar una de nivel N con la «boca» orientada hacia la derecha, por ejemplo, sería:

«Pintar figura nivel N, a derecha:»

Si N es distinto de 0, entonces:

- **Pintar figura nivel N-1, arriba.**
- **Trazar recta hacia la izquierda de longitud lH.**
- **Pintar figura nivel N-1, a derecha.**
- **Trazar recta hacia abajo de longitud lV.**
- **Pintar figura nivel N-1, a derecha.**
- **Trazar recta hacia la derecha de longitud lH.**
- **Pintar figura nivel N-1, abajo.**

lH e lV serían las longitudes de los tramos de unión horizontales y verticales, respectivamente. Como se ve, la definición es recursiva y es necesario disponer de procedimientos para las cuatro orientaciones; además, como todos se pueden llamar a todos directa o indirectamente, es una excelente oportunidad de emplear la definición FORWARD al escribirlos en PASCAL.

A la hora de trazar gráficos con un ordenador, normalmente se considera que

la pantalla está formada por celdillas individuales, cada una de las cuales se puede «apagar» o «encender» (con uno o más posibles colores, según el ordenador), y que se identifican por medio del número de fila y columna que ocupan, estando preestablecida una de las esquinas como origen de la cuenta. La mayoría de las versiones de Pascal permiten manejar esas celdillas o píxels («píxel» viene de «picture element») por medio de procedimientos predefinidos que podrían tener el siguiente aspecto:

Plot (3, 100, 1) «enciende» el pixel de la columna 3, fila 100.
Plot (3, 100, 0) «apaga» el mismo.

o bien

Plot (3, 100) «enciende» el pixel de la columna 3, fila 100.
UnPlot (3, 100) «apágalo».

Incluso es posible que existan procedimientos para trazar rectas entre dos pixels dados:

Draw (1, 5, 100, 80) traza recta de la columna 1, fila 5 a la columna 100, fila 80.

Por otra parte, muchos ordenadores precisan una acción específica sobre el hardware para poder trazar gráficos; para ello emplearemos el procedimiento Modo-Gráfico, que habrá que adaptar a cada caso (o eliminar); utilizaremos el procedimiento Borra-Pantalla para separar claramente las instrucciones necesarias para el borrado de la pantalla, que son dependientes del compilador.

A la hora de trazar las figuras, supondremos que disponemos de un «lápiz» cuya posición en la pantalla será la indicada por las variables H (horizontal) y V (vertical); para mover el lápiz dejando un trazo tras de sí emplearemos el procedimiento Mueve.

```
program Hilbert;

(*-------------------------------------------------------------*)
(* Este programa consta de algunos procedimientos y constantes *)
(* que varían según el compilador utilizado; todos ellos se    *)
(* encuentran adecuadamente señalados.  En concreto, este      *)
(* programa es directamente compilable con Turbo-Pascal.       *)
(* Véase el texto.                                             *)
(*-------------------------------------------------------------*)

const
(*---------------------------------------------------------*)
(* Estas constantes indican las dimensiones de la pantalla *)
(* gráfica; varían de unos ordenadores a otros.            *)
(*---------------------------------------------------------*)

 V_Min = 0;  V_Max = 199;
 H_Min = 0;  H_Max = 639;

var
 H, V,    (* Guardan la posición del "lápiz"   *)
 Ih,Iv,   (* Guardan la longitud de los trazos *)
 N        (* Nivel de la figura a dibujar      *)
      : integer;
 Ca   : char;

(*------------------------------------------------------------------*)
procedure Modo_Grafico;
  (*---------------------------------------*)
  (* Este procedimiento varía con el compilador *)
  (*---------------------------------------*)
 begin
  HiRes
 end;

(*------------------------------------------------------------------*)
procedure Borra_Pantalla;
  (*---------------------------------------*)
  (* Este procedimiento varía con el compilador *)
  (*---------------------------------------*)
 begin
  HiRes;
  HiResColor (11)
 end;

(*------------------------------------------------------------------*)
procedure Mueve (Salto_H, Salto_V: integer);

(* Mueve el "lápiz" el salto indicado, que puede ser positivo o no *)
```

```
begin
  (*--------------------------------------------------------*)
  (* La siguiente instrucción es distinta según el compilador *)
  (* que tengamos.  En algunos casos puede que sea necesario  *)
  (* sustituirla por más de una; véanse los otros ejemplos.   *)
  (*--------------------------------------------------------*)

  (* Traza la recta: *)

  Draw (H,V, H + Salto_H, V + Salto_V, 1);

  (* La posición ha cambiado: *)

  H:= H + Salto_H;
  V:= V + Salto_V
 end;

(*----------------------------------------------------------------*)
procedure ARRIBA  (Nivel: integer); forward;

procedure ABAJO   (Nivel: integer); forward;

procedure DERECHA (Nivel: integer);
 begin
 if Nivel <> 0 then
   begin
    Arriba (Nivel-1);
    Mueve (-Ih, 0);      (* En horizontal hacia la izquierda *)
    Derecha (Nivel-1);
    Mueve (0, -Iv);      (* En vertical hacia abajo *)
    Derecha (Nivel-1);
    Mueve (+Ih, 0);      (* En horizontal hacia la derecha *)
    Abajo (Nivel-1)
   end
 end;


 procedure IZQUIERDA (Nivel:integer);
  begin
  if Nivel <> 0 then
    begin
     Abajo (Nivel-1);
     Mueve (+Ih, 0);
     Izquierda (Nivel-1);
     Mueve (0, +Iv);
     Izquierda (Nivel-1);
     Mueve (-Ih, 0);
     Arriba (Nivel-1)
    end
  end;

 procedure ARRIBA;
  begin
  if Nivel <> 0 then
    begin
     Derecha (Nivel-1);
     Mueve (0, -Iv);
     Arriba (Nivel-1);
     Mueve (-Ih, 0);
     Arriba (Nivel-1);
     Mueve (0, +Iv);
     Izquierda (Nivel-1)
    end
  end;

 procedure ABAJO;
  begin
  if Nivel <> 0 then
    begin
     Izquierda (Nivel-1);
     Mueve (0, +Iv);
     Abajo (Nivel-1);
     Mueve (+Ih, 0);
     Abajo (Nivel-1);
     Mueve (0, -Iv);
     Derecha (Nivel-1)
    end
  end;

 (*----------------------------------------------------------------*)
 function Dos_elevado_a (N: integer): integer;
 var I,J: integer;
 begin
```

```
  J := 1;
  for I:= 1 to N do J:= J * 2;
  Dos_elevado_a := J
 end;

 (*-----------------------------------------------------------------*)


 begin
  Modo_Grafico;
  writeln ('Tras cada figura, pulse Intro para seguir.');

  repeat
    write ('Nivel: '); readln (N);
    if N < 1 then N:= 1; (* no vale nivel menor que 1 *)

    (* Tomamos la longitud de trazo adecuada   *)
    (* para llenar la pantalla con la figura: *)

    Ih:= H_Max  div (Dos_elevado_a (N) - 1);
    Iv:= V_Max  div (Dos_elevado_a (N) - 1);

    (* Cuidamos que no se peguen las líneas: *)

    if (Ih >= 3) and (Iv >= 3) then
        begin
         Borra_Pantalla;

         (* Empezamos en una esquina: *)
         H:= H_Max;
         V:= V_Max;

         Derecha (N);

         readln;
         Borra_Pantalla;
        end
      else
        writeln ('No cabe en esta pantalla.')

  until false
 end.
```

La figura de orden 5 generada por este programa quedaría así:

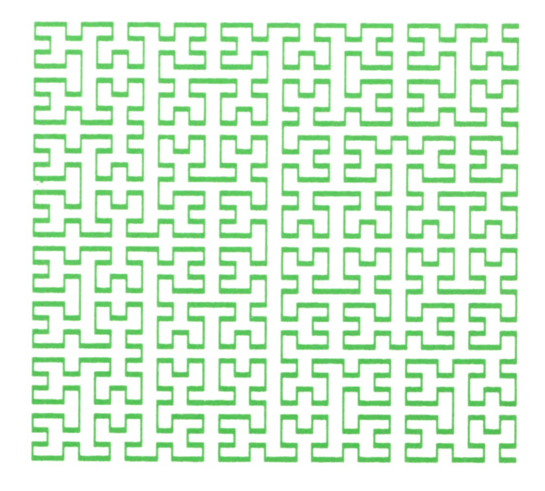




Fig. 2.

Realmente, dependiendo de cuál sea la esquina de la pantalla en que se empiecen a contar las filas y columnas, puede que lo que llamamos arriba sea abajo y viceversa, etc.

Si IH es la longitud del tramo horizontal, la dimensión a lo ancho de una figura de nivel N es igual a 2 elevado a N, menos 1, y análogamente en vertical; por ello se ha empleado la función Dos_elevado_a al calcular las longitudes de tramo.

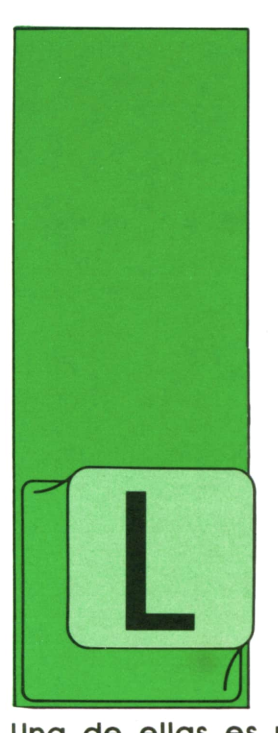
Para terminar, veamos cómo escribir el procedimiento Mueve si sólo se pueden manejar píxeles individualmente, contando con que las rectas son horizontales o verticales exclusivamente:

```
procedure Mueve (Salto_H, Salto_V: integer);
   var I: integer;
 begin
  if Salto_H = 0 then
                                          (* recta vertical: *)
      if Salto_V > 0 then
          for I:= V to V + Salto_V do Plot (H,I,1)
        else
          for I:= V downto V + Salto_V do Plot (H,I,1)
    else
                                          (* recta horizontal: *)
      if Salto_H > 0 then
          for I:= H to H + Salto_H do Plot (I,V,1)
        else
          for I:= H downto H + Salto_H do Plot (I,V,1);

  (* La posición ha cambiado: *)

  H:= H + Salto_H;
  V:= V + Salto_V
 end;
```

OTROS LENGUAJES





Entrada/Salida de datos por pantalla

LOS ordenadores reciben infomación, la elaboran y la muestran nuevamente al usuario.

Existen diversas formas de tomar datos y reflejar resultados. Una de ellas es utilizar el teclado y la pantalla.

La captura de datos desde el teclado se efectúa con la instrucción:

ACCEPT campo

Cuando un programa en ejecución se encuentra con esta instrucción, se para, permitiendo al usuario teclear el dato que se desee, hasta que pulse la tecla <intro>. Los caracteres que han sido tecleados serán el nuevo contenido del campo.

Para visualizar datos por pantalla se debe emplear la instrucción:

DISPLAY { literal / campo }

Este formato muestra en la pantalla un literal numérico o alfanumérico y cualquier campo: teniendo, además, la posibilidad de displayar con una sola instrucción varios literales o variables.

Se ve un ejemplo de estas instrucciones a continuación:

```
IDENTIFICATION DIVISION.
 PROGRAM-ID. EJ-ACCEPT.

ENVIRONMENT DIVISION.

DATA DIVISION.

 WORKING-STORAGE SECTION.

  01 DATO-1                         PIC 99.

  01 DATO-2                         PIC 99.

PROCEDURE DIVISION.

INICIO.
       DISPLAY 'INTRODUZCA EL PRIMER DATO'.
       ACCEPT DATO-1.
       DISPLAY 'INTRODUZCA EL SEGUNDO DATO'.
       ACCEPT DATO-2.
       DISPLAY 'DATO-1 = ' DATO-1.
       DISPLAY 'DATO-2 = ' DATO-2.

FIN-PROGRAMA.
       STOP RUN.
```

En este programa se han definido en la WORKING dos campos numéricos (DATO-1 y DATO-2), de longitud 2.

La PROCEDURE DIVISION está dividida en dos «párrafos», INICIO y FIN-PROGRAMA. Estos se denominan nombres de párrafo y agrupan un conjunto de sentencias. Más adelante se verá la importancia de estos nombres de párrafo, que deben comenzar en el margen A y finalizar en punto, al igual que la instrucción que los precede.

La primera instrucción ejecutable es un DISPLAY que pedirá al usuario el primer dato. Es conveniente, aunque no necesario, poner un DISPLAY antes de un ACCEPT para avisar al operador.

Después se parará el programa hasta que se tecleen los datos y se dé orden de continuación. Los caracteres introducidos se almacenan en DATO-1.

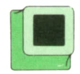
Las dos sentencias siguientes son análogas a las anteriores.

Los dos DISPLAY's siguientes son un ejemplo de cómo visualizar un literal y una variable conjuntamente. La última sentencia del programa es STOP RUN, que lo finaliza.

Todas las instrucciones de la PROCEDURE deben aparecer en el margen B.

Una posible ejecución de este programa podría ser:

```
INTRODUZCA EL PRIMER DATO
3
INTRODUZCA EL SEGUNDO DATO
10
DATO-1 = 03
DATO-2 = 10
```

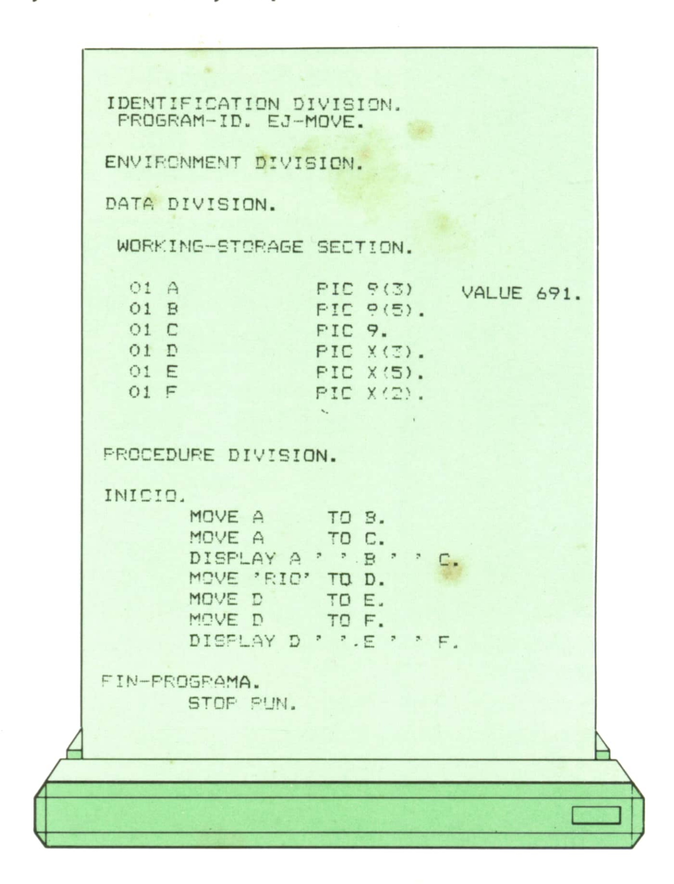


Instrucción de movimiento

El movimiento entre campos o la transferencia de un literal a uno o varios campos se puede hacer empleando la instrucción:



Su funcionamiento se comprende mejor con un ejemplo:



```
IDENTIFICATION DIVISION.
 PROGRAM-ID. EJ-MOVE.

ENVIRONMENT DIVISION.

DATA DIVISION.

 WORKING-STORAGE SECTION.

   01 A              PIC 9(3)    VALUE 691.
   01 B              PIC 9(5).
   01 C              PIC 9.
   01 D              PIC X(3).
   01 E              PIC X(5).
   01 F              PIC X(2).

PROCEDURE DIVISION.

INICIO.
        MOVE A      TO B.
        MOVE A      TO C.
        DISPLAY A ' ' B ' ' C.
        MOVE 'RIO' TO D.
        MOVE D      TO E.
        MOVE D      TO F.
        DISPLAY D ' '.E ' ' F.

FIN-PROGRAMA.
        STOP RUN.
```

En WORKING se definen unos campos. La primera instrucción mueve el contenido del campo A a B y C.

Con el DISPLAY se visualizará: 691 00691 1.

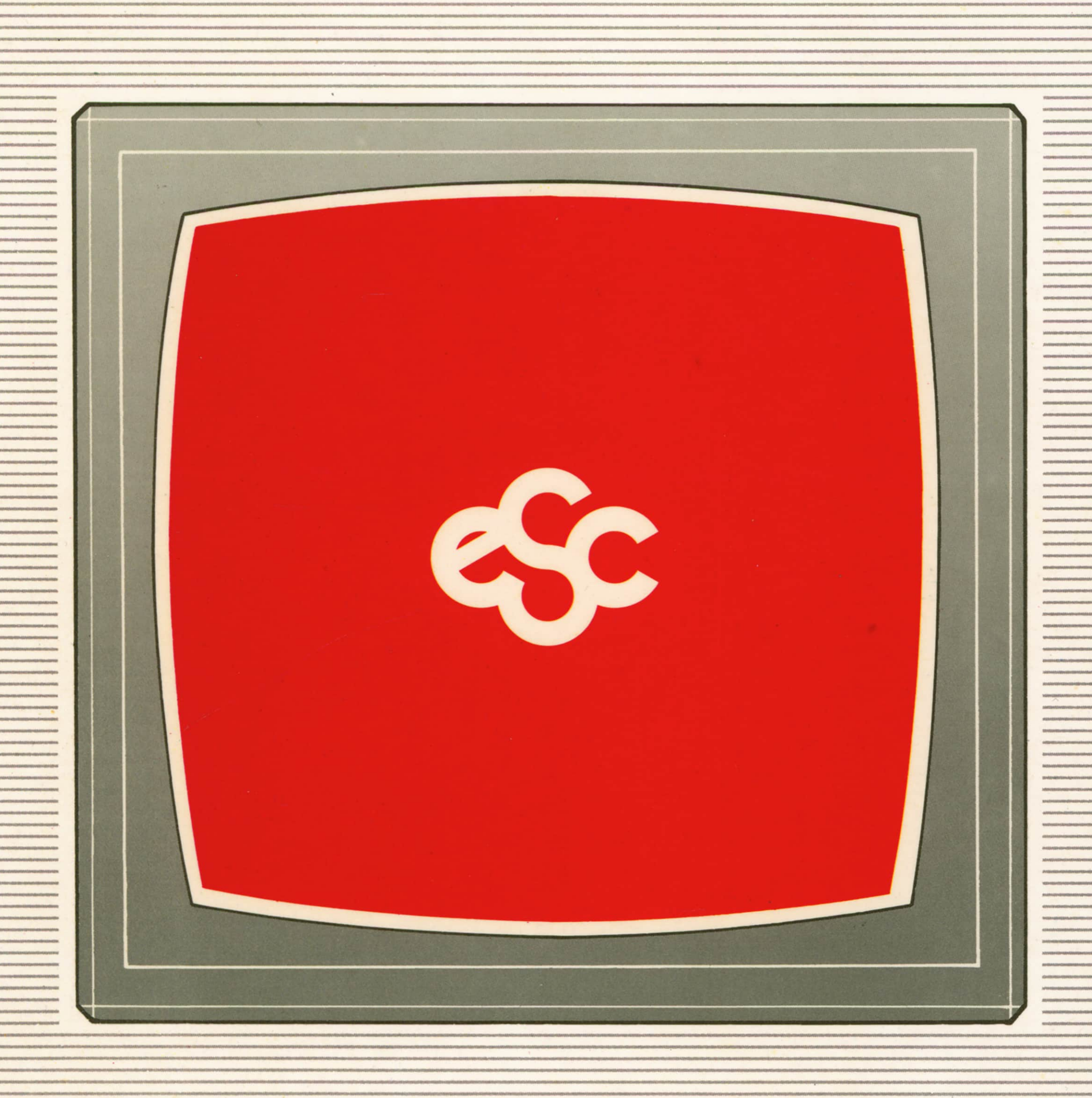
El movimiento de un campo numérico a otro se hace ajustando por la derecha y rellenando con ceros las posiciones libres de la izquierda. Si la longitud del campo receptor es menor se produce un truncamiento y se pierden cifras por la izquierda.

La siguiente instrucción consiste en mover un literal alfanumérico (encerrado entre apóstrofos), al campo alfanumérico D. A continuación, éste se mueve a los campos E y F.

El resultado del DISPLAY que le sigue será: RIO RIO RI.

En el movimiento alfanumérico se ajusta por la izquierda rellenando con blancos por la derecha.

Hay que tener cuidado con las combinaciones de los campos, ya que si el emisor es un campo alfanumérico y el receptor numérico se produce un error durante la ejecución.



▼ EDICIONES ▼ SIGLO ▼ CULTURAL ▼